The Report Committee for Jason Cecil Perkey
Certifies that this is the approved version of the following report:

Wireless Transceiver for the TLL5000 Platform: An Exercise in System Design

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Ranjit Gharpurey

Mark McDermott

Wireless Transceiver for the TLL5000 Platform: An Exercise in System Design

by

Jason Cecil Perkey, B.S.E.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December, 2009

Dedication

To Ellen.

Acknowledgements

The author would like to acknowledge the significant guidance received from Mark McDermott, Ranjit Gharpurey, Adnan Aziz, Eric Quinnell, and Mihir Ravel, as well as the contributions of project members Vanessa Canac, Atif Habib, and Sarat Anumula. Additionally, he would like to acknowledge the material contributions by Analog Devices and VirTex Assembly. Finally, he would like to give special recognition to Freescale Semiconductor for providing the opportunity to pursue this degree.

As well as the above, this work would not have been possible without the vast support from friends, family, coworkers, and fellow-students listed below:

Ellen Perkey

Cohort5 ECD Students – Class of 2009

Chip O'Donnell

Christopher & Rachael Perkey

James & Teresa Perkey

Eric & Leslie Quinnell

Scott Roth

Andrew & Rebecca Short

Daniel & Debra Stenger

Samuel Stenger

and many more…

December 2009

Abstract

Wireless Transceiver for the TLL5000 Platform: An Exercise in System Design

Jason Cecil Perkey, M.S.E.

The University of Texas at Austin, 2009

Supervisor: Ranjit Gharpurey

Co-Supervisor: Mark McDermott

This paper will present the hardware system design, development, and plan for implementation of a wireless transceiver for The Learning Labs 5000 (TLL5000) educational platform. The project is a collaborative effort by Vanessa Canac, Atif Habib, and Jason Perkey to design and implement a complete wireless system including physical hardware, physical layer (PHY-layer) modulation and filters, error correction, drivers and user-interface software. While there are a number of features available on the TLL5000 for a wide variety of applications, there is currently no system in place for transmitting data wirelessly from one circuit board to another. The system proposed in this report is comprised of an external transceiver that communicates with a software application

running on the TLL-SILC 6219 ARM9 processor that is interfaced with the TLL5000 baseboard. The details of a reference design, the hardware from the GNU Radio project, are discussed as a baseline and source of information. The state of the project and hardware design is presented as well as the specific portions of the project to which Jason Perkey made significant contributions.

Table of Contents

List of Tables

List of Figures

PART 1: BACKGROUND

Chapter 1: Introduction

Wireless communication has been an active area of research from the days of Maxwell, Marconi, and Armstrong through the present day research of 4th Generation cellular networks, wireless broadband, and Global Positioning Systems (GPS) [1]. When Motorola first demonstrated the cellular telephone to the Federal Communications Commission (FCC) in 1972, few could have predicted the magnitude of the effect upon modern society. People soon found out that a telephone could be made as mobile as they are, and demand flourished for personal electronic devices that could place phone calls anywhere they traveled. Due to the emerging new market for cell phones and the rise of the Internet in 1995, communication and embedded systems have revolutionized the way society conducts business, performs research, and participates in recreation. Although digital communication has been around since the telegraph was developed in the 1850s, the growth in this area has shown an enormous increase in the past 30 years [2]. Because of this growth, further development of this field provides significant opportunity for research and has inspired the design of a flexible tool for embedded wireless exploration.

This paper will present the hardware system design, development, and plan for implementation of a wireless transceiver for The Learning Labs 5000 (TLL5000) educational platform. The project is a collaborative effort by Vanessa Canac, Atif Habib, and Jason Perkey to design and implement a complete wireless system including physical hardware, physical layer (PHY-layer) modulation and filters, error correction, drivers and user-interface software. While there are a number of features available on the TLL5000 for a wide variety of applications, there is currently no system in place for transmitting

data wirelessly from one circuit board to another. The system proposed in this report is comprised of an external transceiver that communicates with a software application running on the TLL-SILC 6219 ARM9 processor that is interfaced with the TLL5000 baseboard. The details of a reference design, the hardware from the GNU Radio project, are discussed as a baseline and source of information. The state of the project and hardware design is presented as well as the specific portions of the project to which Jason Perkey made significant contributions.

The general motivation is described in Chapter 1, and Chapters 2 and 3 describe background information pertaining to the TLL5000 platform and the GNU software-defined radio. Chapters 4, 5, and 6 elaborate on the early planning information, the system design at hand, and the prototype development respectively. Chapters 7 and 8 conclude with results, future work, and final thoughts.

Chapter 2: TLL5000 Platform

The TLL5000 is the main educational design platform for several circuit design classes at The University of Texas at Austin (UT). Because of its flexibility for embedded systems design and development, it was chosen as the baseboard for the Wireless TLL Transceiver system design project. By adding wireless capabilities to the platform, a powerful design system would be created to allow for innovative embedded systems development.

At the top level, the TLL is simply a baseboard with connections for two separate daughter cards, a programmable FPGA, and a plethora of I/O devices. The 1.5-million gate Spartan3 Field-Programmable Gate Array (FPGA) from Xilinx at the heart of the TLL5000 allows for register transfer level (RTL) hardware design and synthesis. The daughter cards allow for various processors to be connected and programmed to interact with the FPGA and I/Os. In the labs at UT, a TLL-SILC 6219 (hereafter abbreviated TLL6219) is installed in one of the two daughterboard slots with a Linux 2.6.16 kernel running on a Freescale i.MX21 ARM926EJ processor. The baseboard, the TLL5000, is shown in Figure 1.

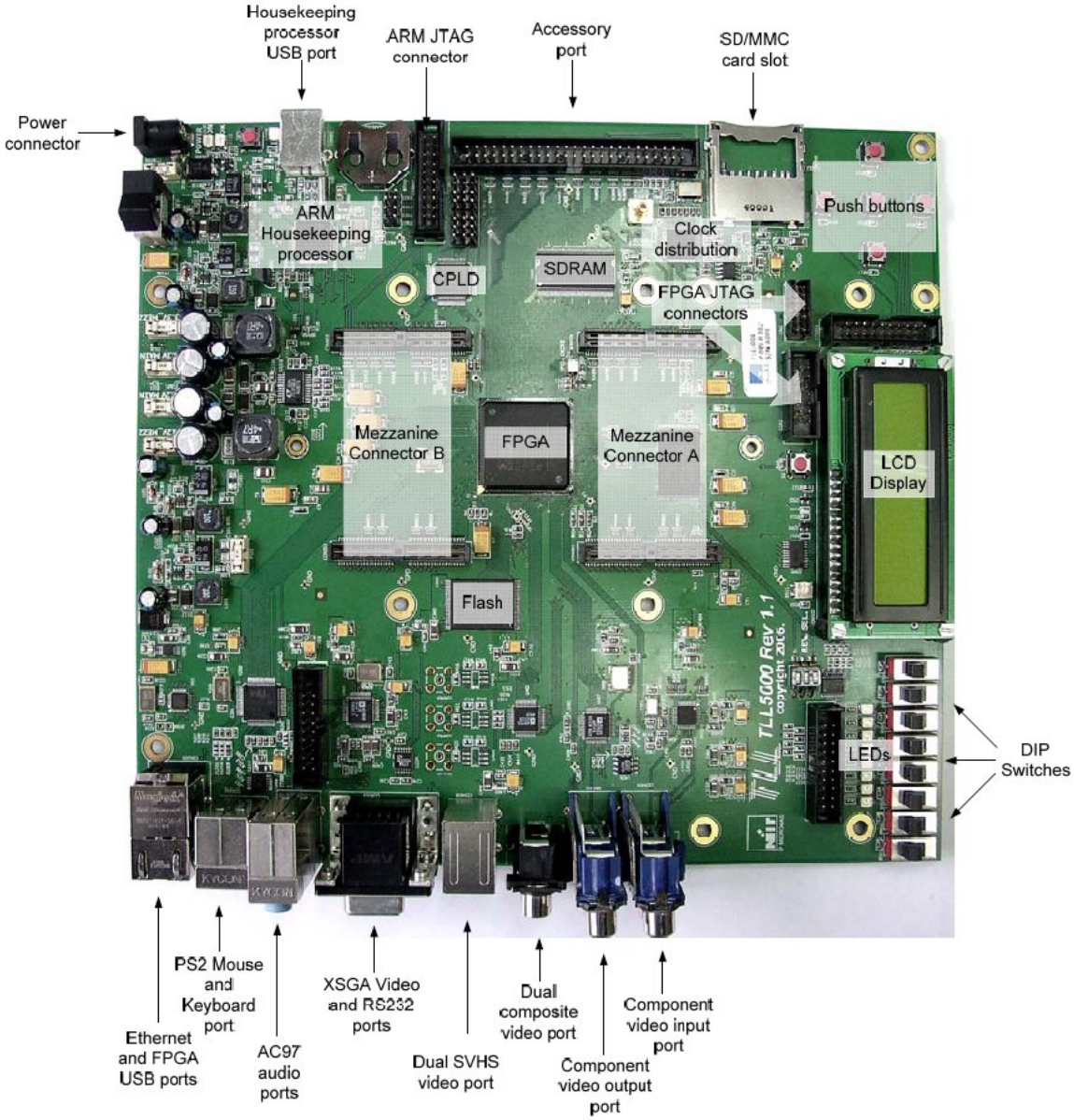


Figure 1. The TLL5000 development system baseboard [3].

The TLL5000 connects to a host PC through a USB port connected to the Housekeeping ARM7 processor [3]. The Housekeeping processor manages the power regulators and power-up sequence for the board as well as controlling the baseboard clock generators. This allows the user to reboot the board and change clocking

configurations from the host PC; which is particularly valuable for debugging purposes if a remote connection to the PC is used, as a hard reset may be the only way to recover the system from certain coding bugs.

The mezzanine connectors allow daughterboards to access the FPGA on the baseboard. In the case of the TLL6219 daughterboard, shown in Figure 2, communication to the ARM9 AMBA bus is routed to the FPGA via a complex programmable logic device (CPLD). This allows an asynchronous protocol to be utilized without the need to pass a clock signal from the processor through the mezzanine connector to the FPGA for communication purposes, and hence reduces the system noise that such a clock would generate [4].

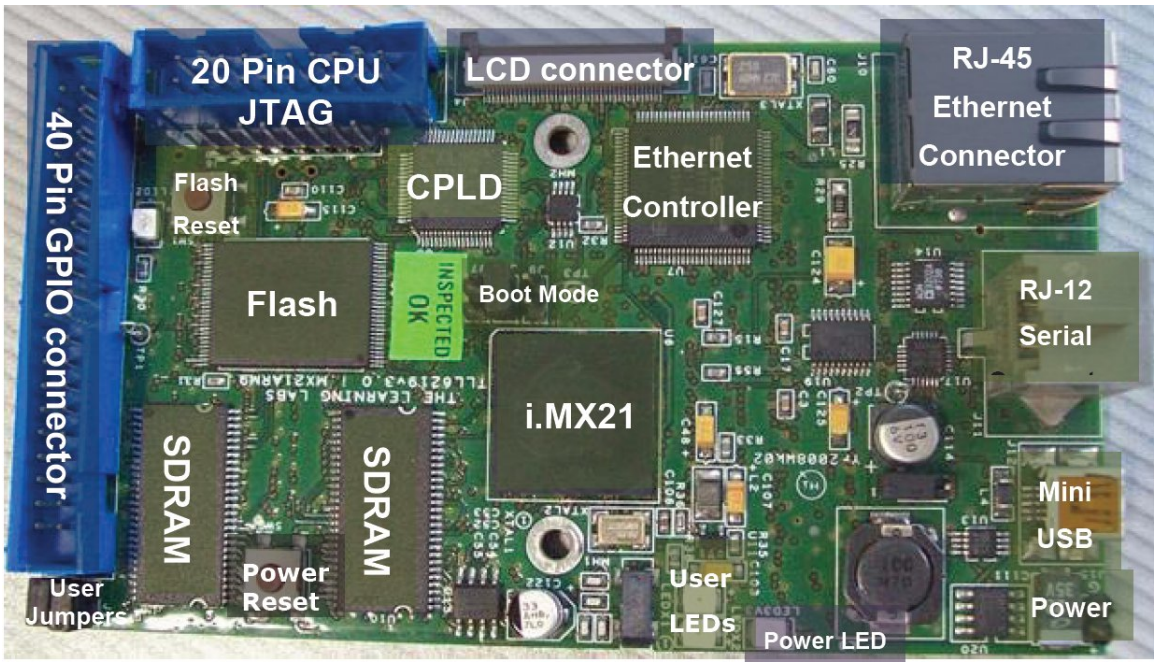


Figure 2. The TLL6219 daughterboard with Freescale i.MX21 processor [4].

Installed on the host PC is a terminal that communicates to the ARM9 processor when it boots into the MicroMonitor (uMON) boot loader through the RJ-45 Ethernet

connector and SMSC Ethernet controller. From uMON, the Linux 2.6.16 kernel can be booted so that the user can execute programs on the ARM9 processor from the host PC [4].

The i.MX21 processor contains six general purpose I/O (GPIO) ports which are multiplexed with other I/O port functionality. On the TLL6219, 36 GPIOs and 4 power/ground rails are routed to the standard 40-pin connector that can easily be accessed via a ribbon cable or individual jumper wires. For example, the i.MX21 contains three configurable serial port interfaces (CSPIs) that can be configured to interface with different types of serial port interfaces (SPIs) present on many types of discrete ICs. These pins can alternatively be set to GPIO pins which can be used to define functionality through the various register settings of the i.MX21 with executable code on the ARM9 [5]. Because of their accessibility and configurability, these GPIO pins can easily be setup for control and debug signals.

For the Wireless TLL Transceiver project, several of the features of the existing baseboard and daughterboard were leveraged. The Spartan3 FPGA was intended for PHY-level algorithmic acceleration on the baseboard, and the i.MX21 processor was utilized as the main general purpose processor (GPP) to run the transceiver software on the ARM9 Linux kernel. Additionally, the 40-pin GPIO accessory port on the TLL6219 was planned for setting control signals on the transceiver as well as programming the discrete IC components through the CSPI ports.

Chapter 3: GNU Radio and USRP

The initial phase of the project included finding a suitable reference design to use as a starting point and as a benchmark. The GNU software radio project was chosen for the similarity of its accomplishments to the goals defined for this project. Both the GNU Radio and the authors' project shared the same objective: to design a flexible wireless transmitter/receiver pair for educational purposes. Although many students have access to wireless design principles and techniques, few educational labs are equipped with a flexible platform to allow physical-layer design with regards to the Open Systems Interconnection (OSI) Seven Layer Model [6]. The GNU software-defined radio project began in April of 2001 with the goals of exploring, defining, and launching an entirely open-source software-defined radio for both students and amateur radio enthusiasts alike [7].

3.1 GNU SOFTWARE-DEFINED RADIO

The general concept behind a software-defined radio is illustrated in Figure 3. An antenna, physical front end, and data converters are required for any physical radio. However, the signal processing that was traditionally performed by discrete semiconductors in physical radios can now become a function of software written for a GPP. In most modern communication systems, this signal processing is typically performed by fixed-function digital signal processors (DSPs) or application-specific ICs (ASICs). The transition of functionality from the hardware domain to a software domain increases flexibility and could potentially allow for decentralized communication networks to replace traditional infrastructure-driven networks [8].

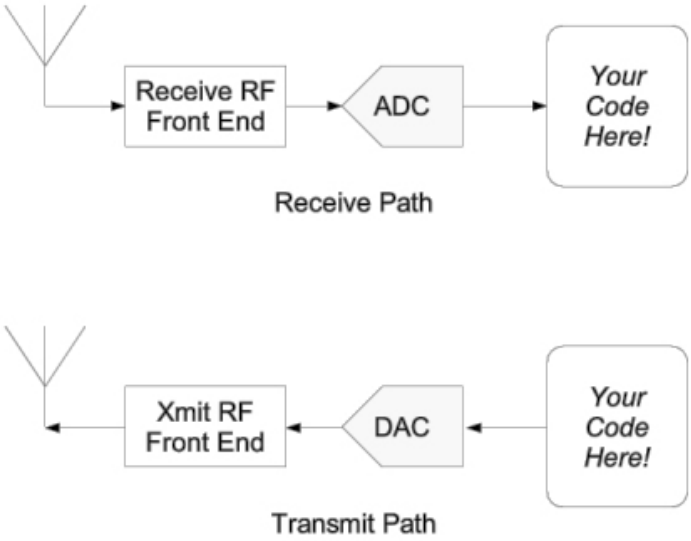


Figure 3. Typical software radio block diagram [8].

3.2 UNIVERSAL SOFTWARE RADIO PERIPHERAL

To facilitate development of the GNU software radio project, a hardware solution was designed by Mathew Ettus and dubbed the Universal Software Radio Peripheral (USRP) [8]. The USRP provides the physical front end for the radio and allows it to communicate with a PC by a standard USB-2.0 connection. Configured for multiple types of daughterboards that can transmit/receive in various ranges of the RF spectrum, the USRP itself contains four 12-bit high-speed analog-to-digital converters (ADCs) that operate at 64 MSPS (mega samples per second) and four 14-bit digital-to-analog converters (DACs) that operate at 128 MSPS. Figure 4 shows the high-level block diagram of the USRP. Assuming the Nyquist frequency is observed, the transmit-side bandwidth could be as large as 64 MHz and the receive-side bandwidth could be as large as 32 MHz.

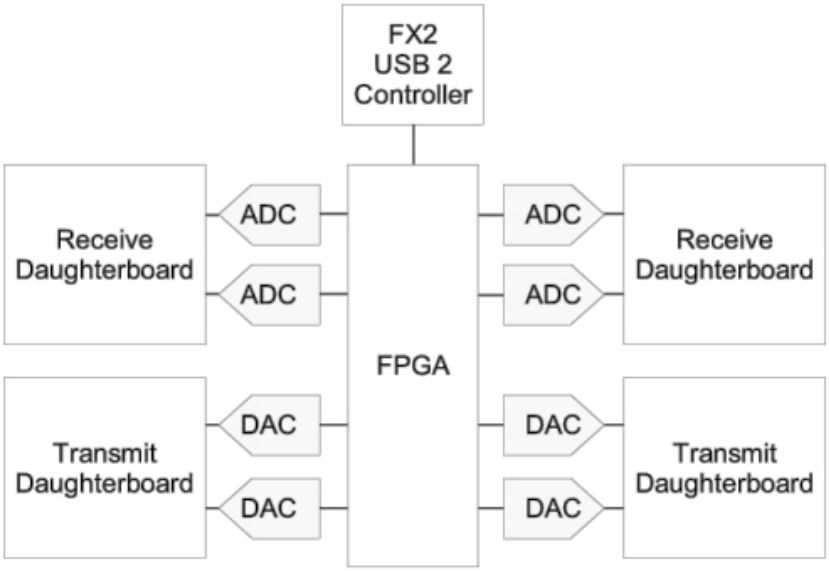


Figure 4. Block diagram of the Universal Software Radio Peripheral [8].

Data is fed from the PC through the USB cable into an FPGA prior to conversion to the analog domain of the daughterboards. Because the FPGA is a programmable medium, the Verilog code written for it is considered part of the software-defined radio, despite being a hardware description language (HDL). The default FPGA code contains a numerically-controlled oscillator† (NCO) and complex multiplier for demodulating the low-frequency complex data into real I (in-phase) and Q (quadrature) signals [10]. The FPGA also contains two 4-stage cascaded integrator-comb (CIC) decimation digital filters and two 31-tap half-band digital filters for signal conditioning in the digital down conversion (DDC) receive path. Figure 5 shows the DDC path through the FPGA prior to entering the Mixed-Signal Front-End Processor (MxFE) by Analog Devices AD9862. The AD9862 contains the ADCs and DACs for the baseboard as well as additional

† The NCO is implemented using CORDIC, an algorithm to compute trignometric functions using only shifts and additions [9].

circuitry utilized for digital upconverter (DUC) transmit path. More details on the AD9862 are presented in Section 3.3.

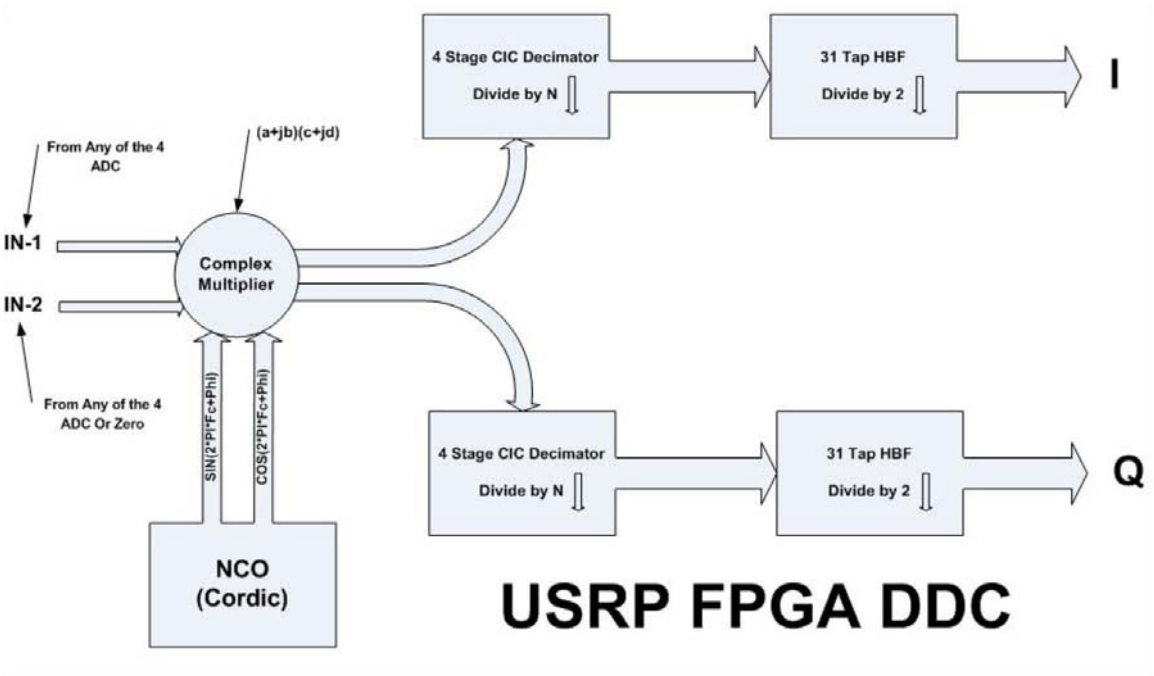


Figure 5. Block diagram of the DDC path [10].

The baseboard has four mezzanine connectors for two transmit daughterboards and two receive daughterboards. Each of the four daughterboards has access to two data-converter channels (ADCs or DACs) for a single, complex I/O or two real I/O channels. The baseboard uses an I^2C interface to an EEPROM on each daughterboard for identification and allows the software to configure itself automatically. Several types of daughterboards are available for the system depending on what type of communication is desired. The Basic Transmit/Receive (TX/RX) daughterboards are intended for system debug, but could also communicate at frequencies lower than 32 MHz with a properly-sized antenna. The Low Frequency TX/RX daughterboards are similar to the Basic TX/RX except their range extends down to DC and also includes a 30 MHz low pas filter

(LPF). The RFX family of daughterboards are designed for a full transceiver system complete with mixers, oscillators, amplification, and filtering. For a complete list of RFX frequencies and a complete list of other daughterboards available, see [11]. The baseboard is shown in Figure 6.

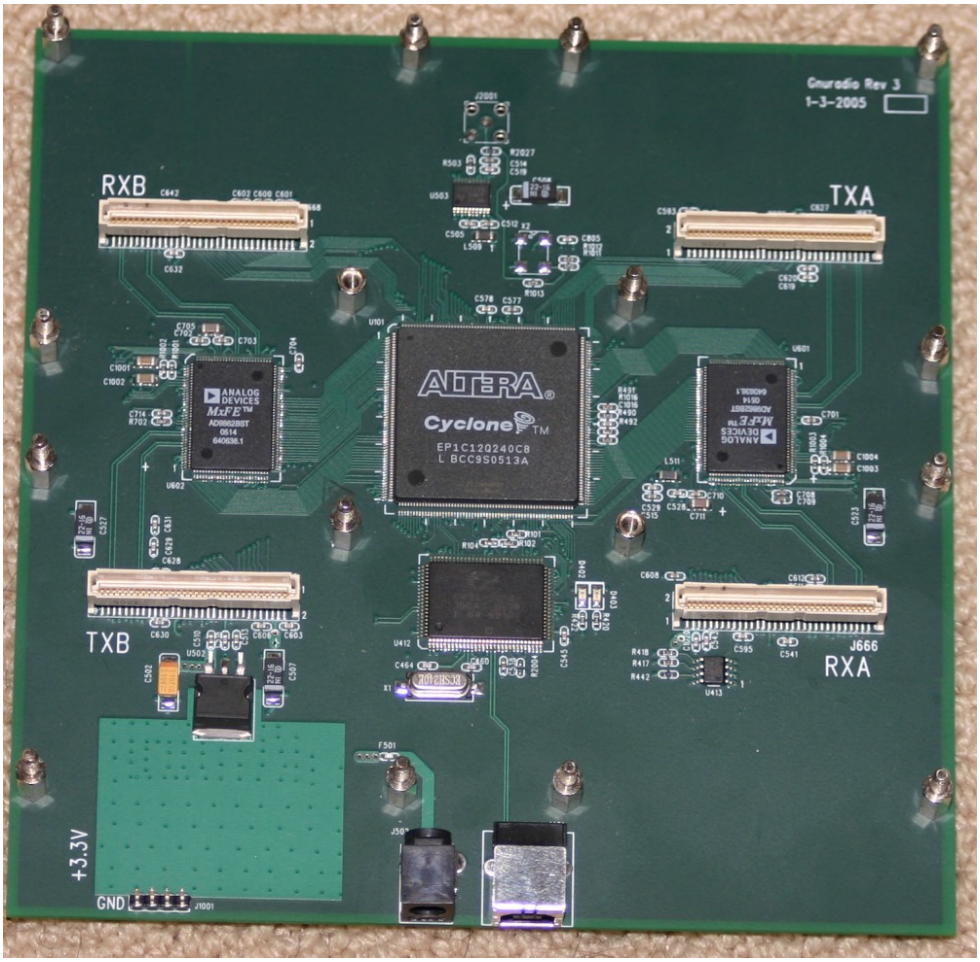


Figure 6. The USRP baseboard by Ettus Research LLC [11].

With such a broad code base and the ability to allow for physical design (in Verilog) as well as software development, the USRP provides an excellent baseline for the TLL Wireless Transceiver project. The open-source design of the GNU Radio and

the USRP had a strong influence on many of the design choices for the TLL Wireless Transceiver hardware. The TLL Wireless Transceiver was designed with the intention of preserving the advantages of the USRP hardware while trying to make additional improvements to some of its shortcomings, described in the following section.

3.3 DESIGN CHOICES OF USRP

The baseboard for the GNU Radio USRP design contains two Mixed-Signal Front-End (MxFE™) processors‡; one for each side. Although this allows separate RX and TX daughter cards to be designed independently (such as the RX only and TX only designs) or together (such as the RXF combined daughterboard), at a system level it necessitates that analog signals be routed through the mezzanine connector. Although this saves cost from an overall system perspective, the increased susceptibility to analog noise is far from ideal. Such connections are notorious for poor signal integrity, mismatched trace lengths, parasitic resistances, etc. In general, sensitive analog connections should be well-shielded to improve signal quality. The Wireless TLL design project will try to avoid such a choice by placing all necessary components on a single printed circuit board (PCB).

The MxFE processor, Analog Devices AD9862, is a flexible mixed-signal front-end processor design for broadband communication applications. It operates at a maximum frequency of 128 MHz and consumes less than 431mA of current at 25^{O}C during full operation [12]. Both the TX and RX paths contain programmable gain amplifiers (PGAs) that can amplify up to 20 dB, and both paths also contain digital Hilbert filters, which are not utilized by the GNU software-defined radio. The RX path consists of two channels of 12-bit 64 MSPS ADCs and decimation filters that can receive

‡ MxFE is a trademark of Analog Devices, Inc. [12].

real, diversity, or I/Q data at either baseband or low intermediate frequencies (IF). The TX path contains two channels of 14-bit 128 MSPS DACs, interpolation filters, and digital mixers that can perform real or complex signal frequency modulation to low IF frequencies. By default, the complex mixers and interpolation filters are utilized as part of the DUC path in GNU Radio implementation. The AD9862 also contains auxiliary ADCs and DACs for system control and monitoring. Two AD9862 processors on the baseboard contain all of the ADCs and DACs for the system. The block diagram is shown in Figure 7.

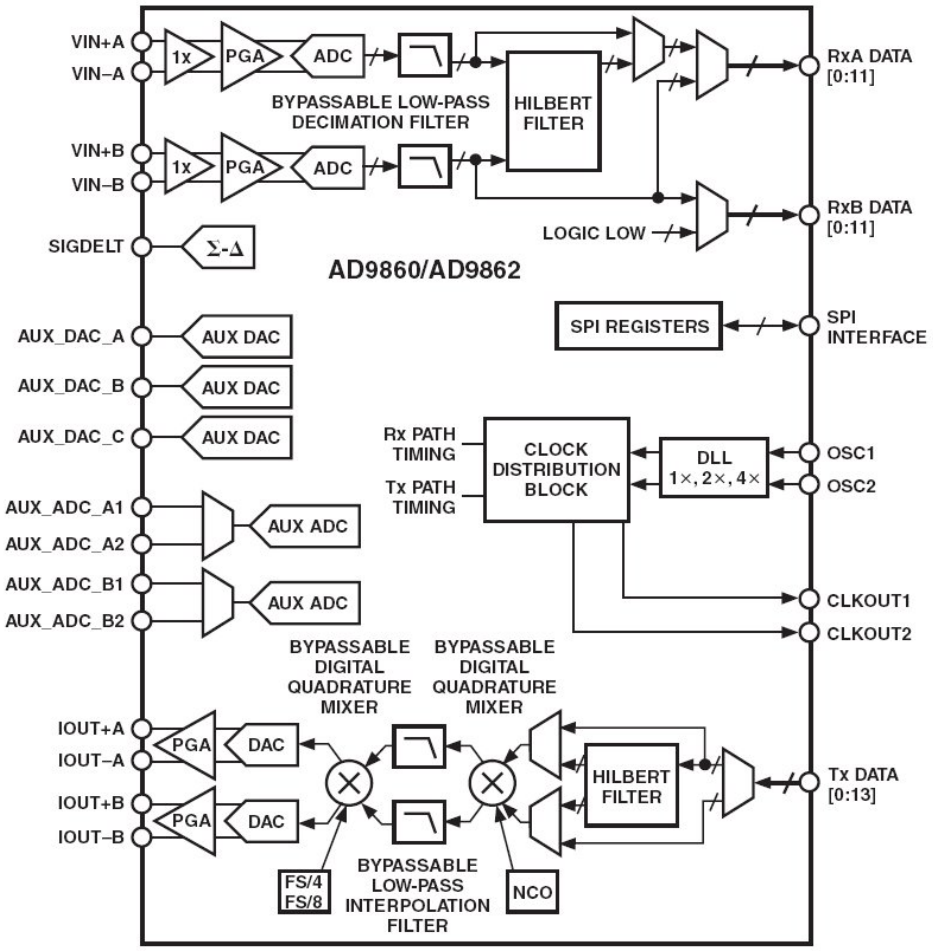


Figure 7. Functional block diagram of the MxFE [12].

The Basic RX/TX daughterboards offer no capability for transmission and reception at frequencies beyond what the MxFE can handle, save through external tuning or mixing. They are primarily intended for system debug and verification. Many of the FPGA signals are brought out to standard connectors suitable for connecting to a logic analyzer. All of the VIN and IOUT signals are individually routed to an SMA connector via a discrete RF transformer.

The RXF daughterboards, the full-transceiver USRP daughterboard design, provided the primary inspiration for the design project. It contains both a transceiver path and an auxiliary RX chain as shown in Figure 8. The RF switch on the transceiver path allows for both transmission and reception on the same antenna. Both the transceiver and the auxiliary RX-only path have their own local oscillators (LO) so that they can tune to independent frequencies. Because the capabilities of the RFX daughterboards are in line with the goals of the Wireless TLL Transceiver, they will provide an excellent starting point for the new design.

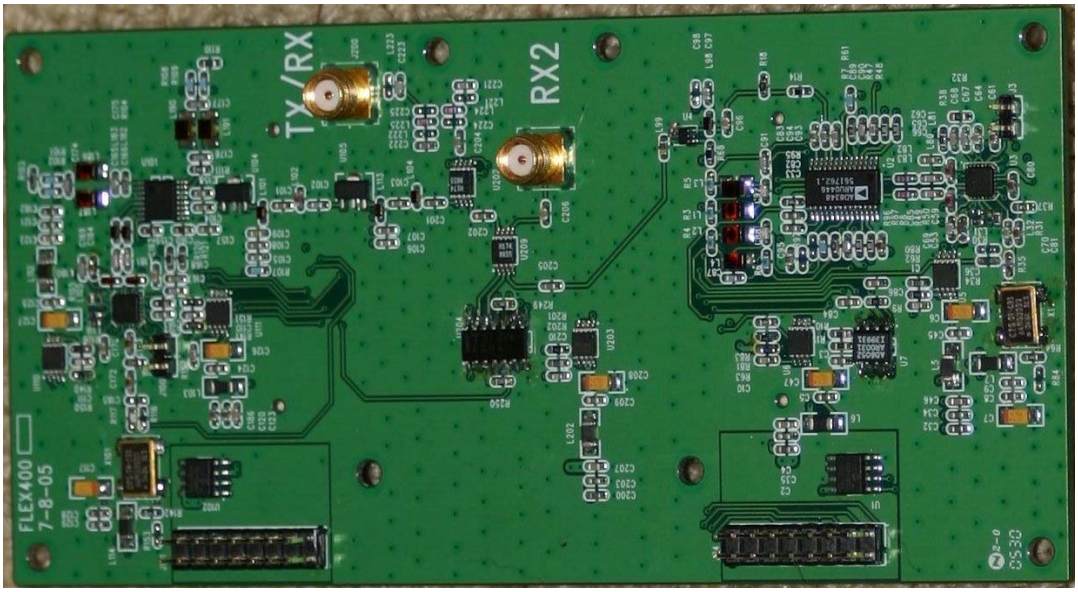


Figure 8. RFX400 daughterboard [11].

PART 2: AN EXERCISE IN SYSTEM DESIGN

Chapter 4: System Planning

Chapter 4 is a discussion of the early planning information and design choices made at the system level prior to beginning the implementation phase of the project. Specifically the Data Link and PHY-layer algorithms are reviewed, as well as the system test plan and the software tools used for the project.

4.1 DATA LINK LAYER SOFTWARE

The plan was to implement the user-interface, transmission protocol driver, and error correction in software running on the ARM9 processor; collectively known in the OSI Model as the Data Link Layer of abstraction. The transmission protocol took inspiration from the 8b/10b coding scheme that combines both commands and redundant information that can also be used for error correction [13].

One low-bandwidth application considered was a single transmitter and multiple receivers, such as in a bus terminal advertisement display. The transmitter would periodically update the images. An application like this would work well within the theoretical bandwidth calculated as well as allowing for bandwidth falling substantially short of the calculations. The likelihood of realizing the initial estimations became more probable when the modulation algorithm was moved from the FPGA to the i.MX21 processor. Thus the protocol employed is designed to handle frequent pauses as the transmitter buffers additional data.

Error correcting codes use symbols containing redundant information to represent the actual message. The presence of the redundant information enables the receiver to recover the original message in the presence of corrupting, physical-channel noise

sources. It was also noted that a code-profiling analysis would need to be performed to determine if the error correction algorithm needs to be accelerated and therefore ported into the FGPA.

4.2 PHYSICAL LAYER ALGORITHMS

In order for the communication system to function properly the data needs to be modified and adjusted to account for various environmental conditions. In the OSI Model, this is known as the PHY-layer or physical layer of abstraction. Data cannot simply be transmitted in its native format with expectations of being received without errors. Such an occurrence would only be possible if there were an ideal "noiseless" channel between the transmitter and receiver, which is not the case in this project. Before it is transmitted, the data should first be translated into a form that is less susceptible to environmental degradation factors. Upon receipt of the signal, the data needs to be decoded from its translated form and then certain restorative algorithms can be applied to recover the original signal's properties. For this project, implementations of binary phase-shift keying (BPSK) modulation and demodulation, equalization, and timing recovery have been developed [14].

It should be noted that BPSK is a subset of quadrature phase-shift keying (QPSK); by setting up a hardware system designed for QPSK, this allows software for both modulation schemes to be developed. Since BPSK only sends data on one phase of the carrier signal, it is the simpler of the two algorithms and can be utilized for debug of the system with fewer complications. By setting up the hardware for QPSK, this allows flexibility for future algorithms to be developed without need for a redesign of the PCB.

In the earliest planning phases, these physical-layer algorithms were intended to be implemented in Verilog on the FPGA. In the GNU Radio design, the modulation and

demodulation was implemented in the AD9862 and the FGPA, while the timing recovery and equalization are implemented in software running on the GPP. A suggestion was made to the authors while still in the planning phases to get all of the physical-layer algorithms running in a C-code model first for the sake of simplicity. The theory is that once a model is running in C, it can be split between software and hardware. Functions that need acceleration for the system to run properly can be ported to the FPGA at a later time.

4.3 TEST PLAN

A test plan was developed for each section of the system, as well as a test plan for the entire system. For the hardware section, a list of specifications to be measured was recorded for the test plan with the intentions to take those measurements when the complete PCB hardware was working. The initial test plan is shown in Table 1.

Specification	Requirements
Receiver Sensitivity (dBm)	Need to get a SMA to SMA loss-control "cable" or test fixture
Power (mW)	Need a standard multimeter to measure used Amps and check the Power supply voltage
Bit Error Rate (bit/sec or dB)	Need 5000+ samples to measure with and without ECC enabled. Can take measurements at different distances / transmit powers
Noise Figure Measurement	Not sure how to measure this... Seek advise; potentially can ground inputs to the transmit section, turn all of the variable amplifiers to maximum, and measure the output signal.
Latency	Measure average Latency through the system in loop back mode by setting interupts to fire at first reception of a packet
Maximum Bandwidth (Mbps)	Enter a continuous transmission mode and check the Maximum Bandwidth matches (less some overhead) the calculated theoretical Bandwidth
Distance - how far can we transmit?	Need one mobile station (set up a PC on a cart) while one can stay in the lab. Hop from outlet to outlet unless I can determine a portable Power supply (backup UPS maybe? or install software on a laptop)
Other Considerations for Debug	
Develop Verilog test routine to transmit and receive a small data set (~20 entries) or one value at a time f	
How do we check the board components to see if the components are working properly?	
Where do we Need testpoints on the board?	
What signals do we Need to send to the aux ADCs?	

Table 1. Original hardware test plan.

Receiver sensitivity, maximum bandwidth, and bit error rate (BER) were the measurements of primary interest for judging the quality and performance of the newly developed system. Using a loss-adjustable connection between boards or in a loop-back connection on a single board would allow for receiver sensitivity measurements. For maximum bandwidth calculations, ideally it would be measured with two systems connected by a noiseless cable such as a well-shielded co-axial cable. BER measurement can be taken in multiple configurations both with and without the error-correction code (ECC) algorithm enabled. By quantifying the BER at varying distances, the effectiveness and improvement due to ECC can be properly characterized.

4.4 HARDWARE DEVELOPMENT TOOLS

One topic that may not be obvious to those setting out to study an open-source design is the issue of tools. When the USRP design was chosen, it was because of the open-source nature and the knowledge that all of the technical details are posted for the public. While this is a true statement, information posted online may not always be in a format that is readily available. The schematics for the USRP are all posted both in PDF format (easily readable for everyone) and in the native format of the tool they were generated with, gschem (part of the gEDA open-source toolset). There were some intentions initially to reuse some of the existing schematics, however that proved to be a challenge. For example, the fact that the schematics were generated using gEDA is not initially obvious from the documentation. This complication was discovered after an extensive search of the online resources available.

Gschem is an open-source, freely available schematics editor that is included in the gEDA package. This package as a whole seems to be developing slowly, despite some of the excitement over the project in both 2004 and 2007 [15] [16]. One PCB-

layout and fabrication team considered for the project did not offer support for the gEDA toolset. With the discovery that the schematics would need to be redrawn in a different format prior to layout, little additional effort was spent learning the intricacies of the suite. That being said, it can still be downloaded and installed, but the user-interface is cumbersome for novice users. Although the author searched extensively for a compatible PCB layout tool, nothing was found that could read or convert the open-source gschem format. There was a conversion tool included in the gEDA package that appeared on the surface to perform this task. However, after several attempts with this course of action, it was abandoned after conversion/importing failed. Ultimately all free tools were abandoned for lack of usability and ease of learning-curve.

The physical design of the USRP is posted in Mentor PADS format. The free viewer was tried, but the format of the USRP artwork seemed to be incompatible with the free viewer available from Mentor. Fortunately another concurrent project at UT acquired a license for the full version of PADS which was able to view the layout of the USRP (specifically the daughter cards) as posted. Several free viewers were explored prior to this acquisition, but only one, CAM-350 from DownStream Technologies, was successful in viewing portions of the layout. Ultimately, all of the tools solutions utilized did not come from open-source projects.

For final schematic design, a short-term license of Cadence OrCAD was made available by Freescale Semiconductor. This turned out to be a realizable solution; OrCAD had a built-in tutorial, excellent documentation and an extensive library of components such that PCB design was finally feasible. The tutorial allowed an unfamiliar user to quickly become proficient with the software. Because the library already contained symbols for the discrete ICs utilized, the creation of schematics was

not nearly as time-intensive as it would have been with previously-considered toolsets. The complete set of schematics generated using OrCAD are included in Appendix C.

Although it was understood at the beginning of the project that the lack of a known tool suite was a problem that needed to be solved, this area of the project was greatly underestimated. Much focus and energy was placed into open-source tools early in the project and unfortunately cost many man-hours that could have been spent on more useful academic exercises.

Chapter 5: Design of a Wireless TLL Transceiver

Chapter 5 is a discussion of the hardware system design choices from the implementation phase and the resulting schematics. Presented are the high-level design choices, the overall plan for the design, and the details of the discrete components selected for the hardware.

5.1 DIGITAL MODULATION VS ANALOG I/Q

On the RFX design, the I and Q channels of the QPSK signal are separately routed on the PCB from the MxFE to the mixer. This technique can lead to phase mismatch noise and differing levels of interference on each of the channels. Analog mixing was required before advances in digital signal processing made digital mixing methods a viable option. A more modern technique was desired for the present design where I and Q channels are mixed digitally and passed through a single ADC. The single analog channel would eliminate the need for careful length and impedance matching of separate traces.

Unfortunately, the decision to use digital mixing instead of the techniques employed on the USRP would require reconstructing almost the entire design, leaving little similarity to the reference design. This lack of similarity meant that additional time would need to verify functionality and also that board layout would have to proceed from scratch. Although the original intent of the project was stay as close to the reference as possible, the risk associated with trace mismatch was considered very high. Also, taking the digital-mixing course of action was known to diminish the chances of completing the PCB design project prior to graduation. Despite these downsides, it was decided that the

digital modulation technique would be more academically fruitful and of greater benefit to potential future work on this project.

5.2 ISM BAND SELECTION

When designing any wireless system, the range of frequencies for operation should be defined and known at the project outset. Since the intended scientific application is for low power lab use only, one might argue that any frequency could be chosen at random. Although this may be theoretically correct in an isolated lab free from RF noise, in practice there are a host of operating rules and many devices bleeding their own electromagnetic radiation. In the United States, the FCC regulates the spectrum allocation, while other countries and continents have their own separate organizations for such regulation. Regulation is beneficial as it prevents unknown objects from creating interference that adversely affects commercial wireless products.

The Industrial, Scientific, and Medical (ISM) bands were allocated by all the regulatory bodies for use in academic projects with international recognition to facilitate operation in global collaboration [17]. Table 2 shows the frequency bands available to the ISM community.

Center frequency	Bandwidth (MHz)	Availability
6.780 MHz	0.030	Subject to local acceptance
13.560 MHz	0.014	
27.120 MHz	0.326	
40.68 MHz	0.040	
433.92 MHz	1.740	Region 1 only*
915 MHz	26	Region 2 only**
2.450 GHz	100	
5.800 GHz	150	
24.125 GHz	250	
61.25 GHz	500	Subject to local acceptance
122.5 GHz	1000	Subject to local acceptance
245 GHz	2000	Subject to local acceptance

*Region 1: the Americas, Greenland and some of the eastern Pacific Islands.
**Region 2: Europe, Africa, the Middle East west of the Persian Gulf including the former Soviet Union and Mongolia.

Table 2. The ISM bands of spectrum allocation [17].

Notice in the table above that there are three primary bands of interest: 433 MHz, 900 MHz, and 2.4 GHz. Frequencies below 433 MHz would offer less than 1 MHz of bandwidth. Above those frequencies, at 5.8 GHz and higher, the sensitivity to PCB design parameters would be too great for a previously un-trained group of students. At 2.4 GHz, many components are available, but commercial Wi-Fi and Bluetooth access is allowed in this band, and as a result it has a great deal of noise [18]. At 900 MHz, there is much activity in the United States due to cordless phones, hand-held personal radios, and other unlicensed devices. To avoid both of these distinct sets of RF interference, the 433 MHz band was chosen to plan the Wireless TLL Transceiver design around. Note that this would restrict the available bandwidth to 1.74 MHz if used outside the US. However, within the US there is an allocation of bandwidth from 420 MHz to 450 MHz that allows for a very high-performance system.

5.3 SYSTEM BLOCK DIAGRAM

A survey of RF semiconductor components was taken to search for a collection of products that would complete complimentary RX and TX paths for the Wireless TLL Transceiver. To collect and review the new system of discrete ICs, a system block diagram was mapped out for the entire system. The entire list of products that were reviewed to develop this block diagram would be impractically large to discuss. Those that were selected have their primary block diagrams displayed in the figures in this section; larger versions of these block diagrams are available in Appendix B.

On the RX side, the signal on the antenna enters the RF-In port of the MC48S803 silicon tuner where it is amplified by the variable-gain LNA before being upconverted to the first IF for noise filtering. The signal is then downconverted to a second IF for additional filtering and amplification. See Figure 9 below.

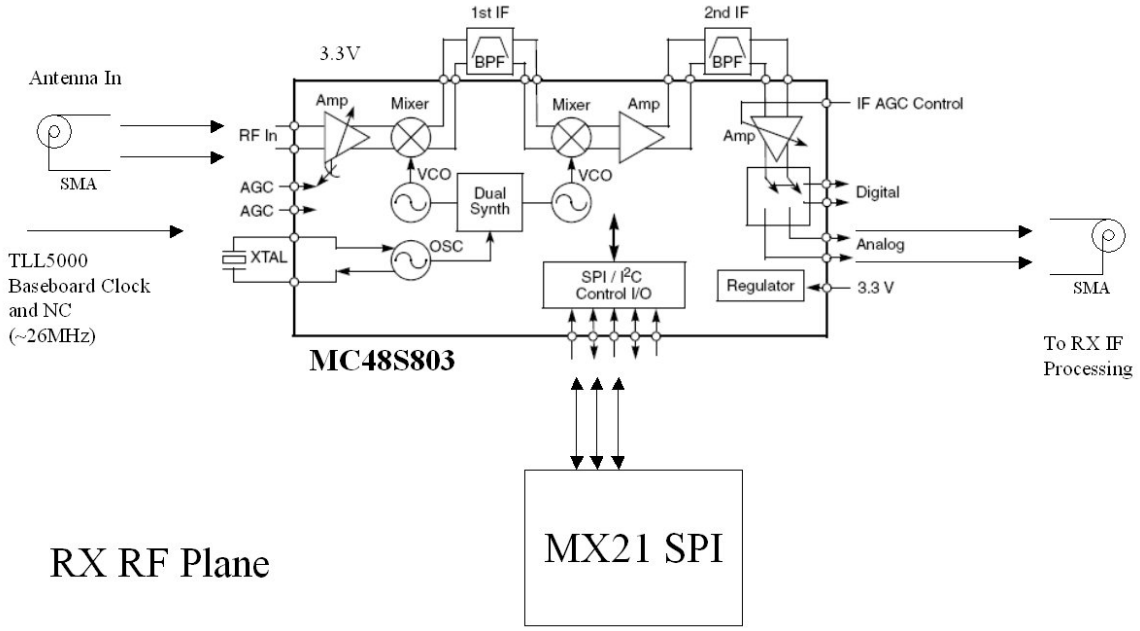


Figure 9. RX RF Plane [19].

On the PCB, the signal then travels through SMA connectors and a shielded coax cable to the AD6653 RX processor on the RF IF plane of the PCB. Immediately upon entry into the AD6653, the signal is conditioned by the sample-and-hold amplifier (SHA) for digitization by the ADC. The digital RX signal is then mixed with offset carriers for separation into I and Q channels. The separate results are passed through a series of digital filters for rejection of the negative image before coarsely being upconverted back to a low-IF for output buffering. The resulting 12-bit I and Q signals can then be sampled on alternating clock cycles by the FPGA for PHY-layer processing as shown in Figure 10.

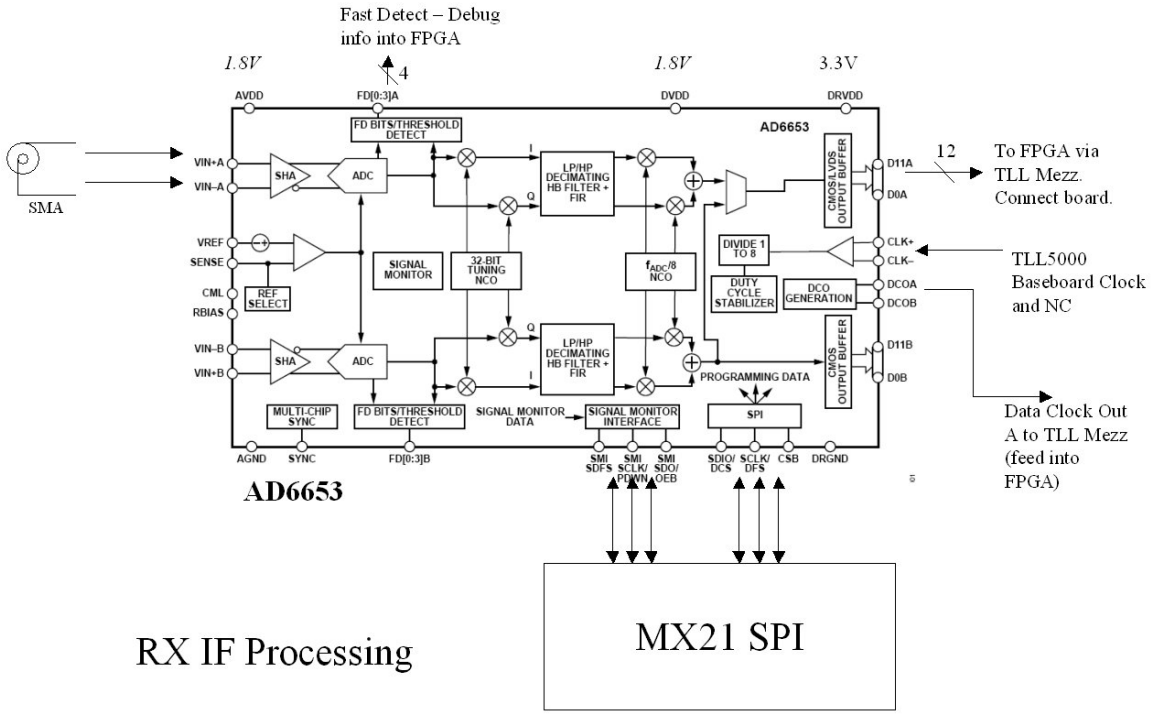


Figure 10. RX IF/Baseband Plane [20].

On the TX side, the digital 14-bit signal generated from the FPGA is sent first to the AD9857 for filtering, interpolation, and digital quadrature-modulation into separate I and Q channels. The output can then be scaled to maximize the range of the 14-bit DAC before it leaves the AD9857 in its analog, low-IF form. To add additional amplitude control to the TX signal, it is then passed to the AD8375 variable-gain amplifier (VGA). See Figure 11 below.

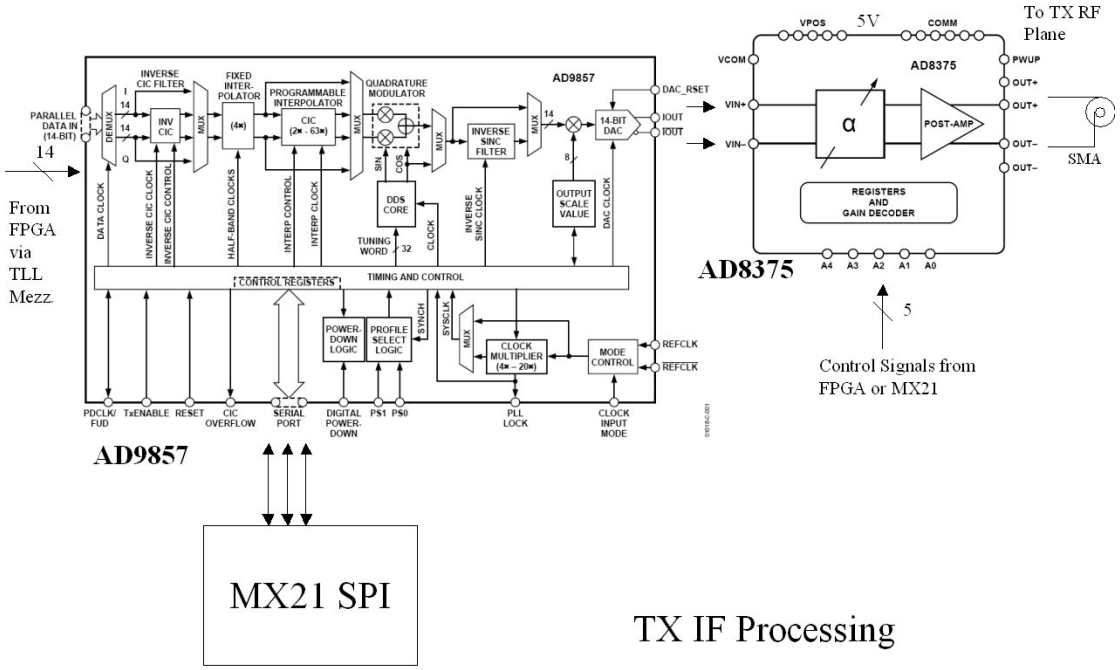


Figure 11. TX IF/Baseband Plane [21] [22].

The signal then travels from the TX IF plane of the PCB to the TX RF plane through a shielded coax cable. At the RF plane, the AD8343 mixer modulates the signal to its final RF frequency. The high-frequency signal is then boosted by a gain-block amplifier and a power-amplifier (PA) before being finally delivered to the antenna as shown in Figure 12.

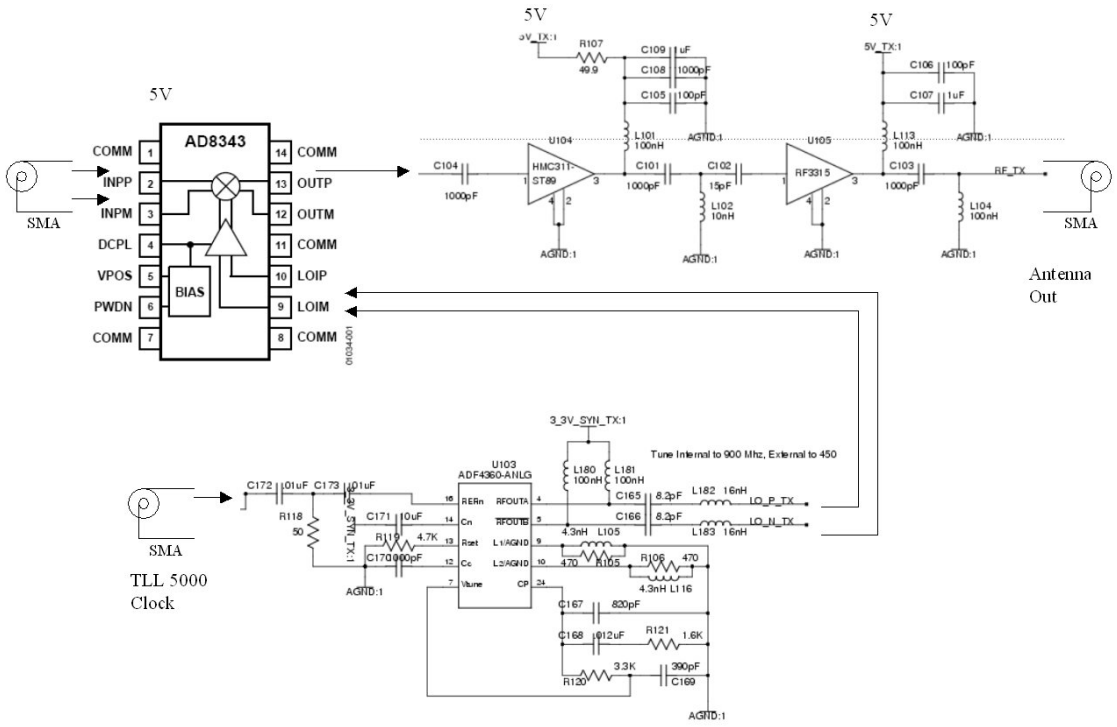


Figure 12. TX RF Plane [11] [23].

5.4 PLANNING PCB FLOORPLAN

To facilitate easier PCB design and to prepare for PCB fabrication, a floorplan was devised for the final PCB. This floorplan incorporates the devices from the system block diagram, and is a first pass at physical space allocation. The isolation areas are key for noise separation and/or reduction, and with the proposed floorplan the single PCB could even be physically separated if further separation was required as shown in Figure 13.

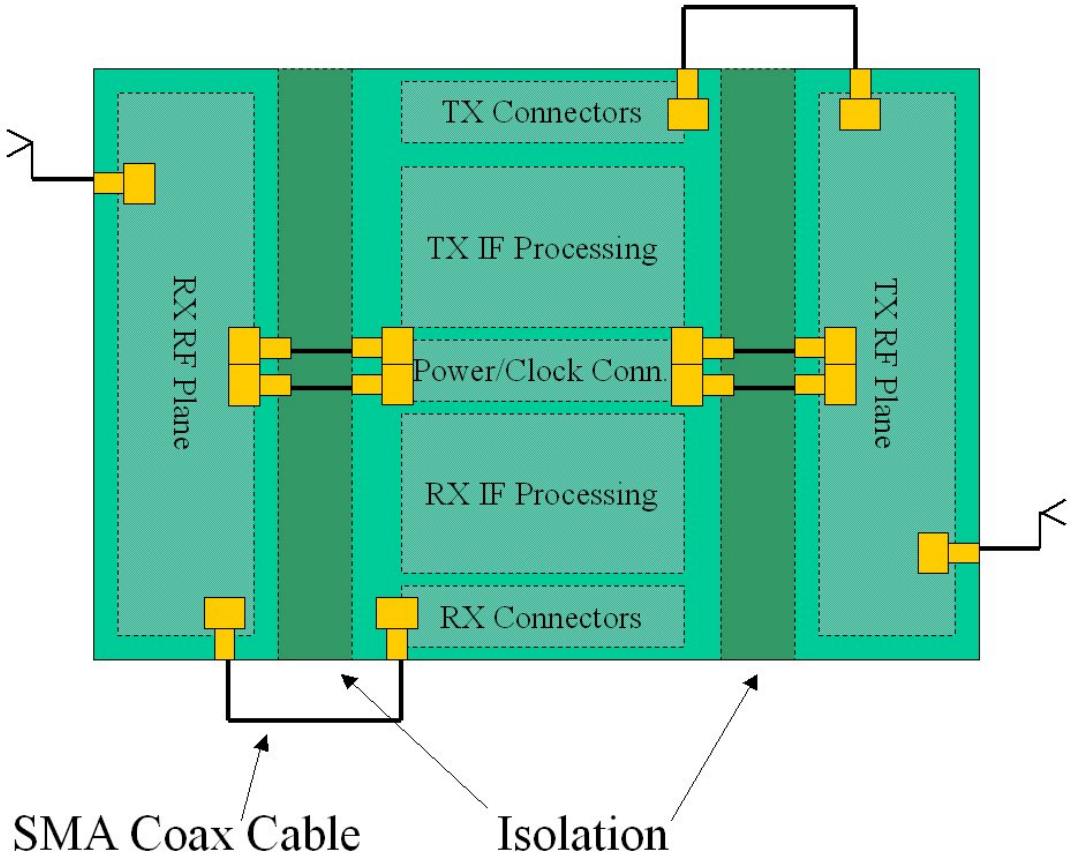


Figure 13. High-level floorplan for the PCB design.

5.5 BASEBAND AND IF COMPONENT DETAILS

The AD9857 quadrature digital upconverter from Analog Devices was selected as the baseband transmitter-side processor because of its high-bandwidth and flexibility of operation. The internal clock rate can operate up to 200 MHz with a 14-bit data path sampling at up to 200 MSPS [21]. It includes a direct digital synthesizer (DDS) to drive the quadrature modulator, an inverse CIC filter, and both fixed and programmable interpolators. The three modes of operation allow the AD9857 to operate as a quadrature

modulator, an interpolating DAC, or as a fixed-frequency wave generator in single-tone mode.

The Analog Devices AD6653 IF diversity receiver was selected as the baseband receiver-side processor because of its similarity to the TX processor in bandwidth and flexibility that would create symmetry within the design. Depending on the speed-grade purchased, it can sample at up to either 125 MSPS or 150 MSPS. Although there are two RF input channels, the present design will only utilize one. The ADC output is connected to the digital downconverter (DDC) that is controlled by a 32-bit NCO [20]. The received signal also goes through a decimating half-band filter and a fixed finite impulse response (FIR) filter before exiting through the 12-bit CMOS or LVDS output.

The AD8375 ultralow distortion IF VGA from Analog Devices was selected as the baseband VGA for both the RX and TX paths. The range of gain from –4 dB to +20 dB make it an excellent match for our system as it is difficult to predict how much amplification (or possibly attenuation) may be required *a priori*. The 1-dB gain steps are digitally controlled by a 5-pin parallel digital input [22]. The simple control makes it easy to incorporate into the design, and it also has a noise figure of 8 dB at maximum gain.

5.6 RF FRONT END HIGHLIGHTS

To start the search for RF components, a goal was set to find a single-IC solution to reduce the number of components. In general, the fewer number of discrete ICs reduces the potential for errors in PCB design. On the RX side of the RF front end, a single IC was found that appears to complete the entire path, the MC48S803 silicon tuner from Freescale Semiconductor. For the TX side, no such single-IC solution was located.

Thus a suitable mixer was located to combine with the TX path of the USRP reference design. This allows for a very small amount of reuse from the original plan.

The low power CMOS broadband tuner, MC48S803, is able to cover RF inputs from 48 MHz up to 1 GHz [19]. These inputs can then be mixed down to an IF between 30 MHz and 60 MHz. The double-conversion architecture first up-converts the received signal to a higher IF and passes it through an external band-pass filter before down-converting to the desired lower IF where it passes through a second external band-pass filter. There is adjustable gain at both the RF input and the second IF, as well as fixed amplification between the two IF stages. Typical noise figure performance is 7.0 dB, and the chip includes two frequency synthesizers for the internal mixers.

The TX plane includes the AD8343 mixer from Analog Devices as well as a repeat of the components present in the transmitter of the RFX400 USRP daughterboard design. The ADF4360 from Analog Devices provides the LO frequency synthesis to the mixer. The RF signal output from the mixer is amplified first by the HMC311 gain-block amplifier from Hittite Microwave and then passed to the RF3315 power amplifier from RF Micro Devices for final amplification to the antenna.

5.7 POWER SUPPLY FOR PCB

When it came time to power the PCB, the authors chose not to re-invent the wheel. Since the reference design already had a working and robust power supply design, the natural two options were to either power the new PCB from the connections to the motherboard's power grid or to copy that same power supply architecture with minimal modifications to independently power the new PCB. The primary considerations were power supply current demands, noise transmission, and cost. The cost of a secondary power supply to a small lab setup is almost irrelevant, so this did not

cause any concerns. A rough order-of-magnitude calculation shows the potential demand on the power supply could easily exceed 2 Amps; see Table 3 below. This might push the limits of any ribbon cable connection that was planned to be connected to the motherboard setup. Noise is always better if a separate power supply can be utilized, and it is highly desirable to have a low-noise power supply for the RF components. Because of all of these factors, the re-use of a previously proven architecture was selected.

Device	Type	Current Draw (mA)	Temperature / Notes
AD9857	TX	615	25C
AD6653	RX	809	Full temp. range
AD8375	VGA	150	Full temp. range
MC1234	Tuner	315	Typical current, 25C
AD8343	Mixer	75	Full temp. range
ADF4360	Freq. Synth.	37.5	25C
HMC311	Amp	74	25C
RF3315	PA	170	Full temp. range

Table 3. Maximum specified current draws for major components.

The main power supply for the TLL5000 is designed around the LT1933 step-down switching regulator from Linear Technology. Based on this starting point, the necessary modifications were made to include a 5-V, 3.3-V, and 1.8-V power supply. For complete power supply schematics, see Appendix C. Note that in the schematics, only the 3.3-V power supply is shown. From the equations in [24], the component-value changes are shown in Table 4. Power supply components for each of the desired rails are required to create the additional power supplies.

	Schematic Label		
	R7	L9	C18
Vout (V)	R (kΩ)	L (µH)	C (µF)
1.8	4.5	11	33
3.3	16.5	18.5	18
5	30	27	12

Table 4. Power supply component values.

Chapter 6: Prototype Development

Chapter 6 is an overview of the prototype development for verification of the present design. The prototype methodology, a breadboard implementation, and all of the necessary elements for the breadboard are discussed. The software developed for hardware verification and lab setup are also detailed.

6.1 BREADBOARD IMPLEMENTATION

The original intent for the project was to develop a PCB layout and verification by doing a first pass design, full review, and then proceeding to fabrication. This is the standard way most PCB designs for small-scale projects are done within the industry [25]. Because PCBs are relatively cheap and most of the cost goes into the design of the PCB, making revisions can cost incrementally very little [26]. Research into simulation methods for the entire PCB system yielded no plausible options. At the point in the project where the design diverged significantly from the reference, a suggestion was made from several sources to attempt a solderless breadboard implementation.

To facilitate prototyping efforts, breadboard adapters were sought so that the discrete packaged devices could be soldered onto a form factor that can easily plug into a solderless breadboard. The advantage of solderless breadboard prototyping is connections can be modified quickly and easily to change design parameters with little danger of causing irreversible damage to a PCB or packaged device. It also allows for a systematic debug and bring-up approach. For example, digital circuitry can often be booted and tested while analog circuits are left grounded or floating to remove additional variables from the list of what might possibly be causing issues.

6.2 ATX POWER SUPPLY REUSE

To setup a prototype environment, it was decided that an external power supply would be easier to control and monitor for debugging purposes than using the regulators designed for the PCB. This would allow for additional current measurements and fuses to protect the ICs. Two Dell ATX power supplies were at the disposal of the team from out-of-date desktop computers, so this seemed like an economical and natural first choice.

While there are many guides readily available on the Internet that describe such power supply salvaging techniques, the manufacturer's website ended up being the most useful. Although the ATX spec, which was first published in the early 1990s [27], is commonly used, Dell did not follow the standard wiring guidelines for their motherboards and power supplies for several years. As such, initial attempts to follow standard startup instructions were unsuccessful. However, the issue was resolved by consulting the manufacturer's website where the appropriate wiring diagram was posted [28].

6.3 BREADBOARD ADAPTERS

Breadboard adapters for the baseband half of the PCB were researched and purchased as a starting point. This includes the AD9856 transmitter, the AD6653 receiver, and the VGA AD8375 (shown in Figure 14 and Figure 15). Note that the packages for the AD6653 and the AD8375 both have a backside connection for thermal conductivity as well as to minimize resistance on the ground connection to the PCB. Modifications to two of the adapters were performed before soldering so that the backside ground connectors could be accessed through an additional soldered wire. Although this is not as ideal as the connection that would be present on a PCB, this

connection is more desirable than not having one at all. Also, in the case of the AD6653 receiver, this was the only package connection to analog ground.

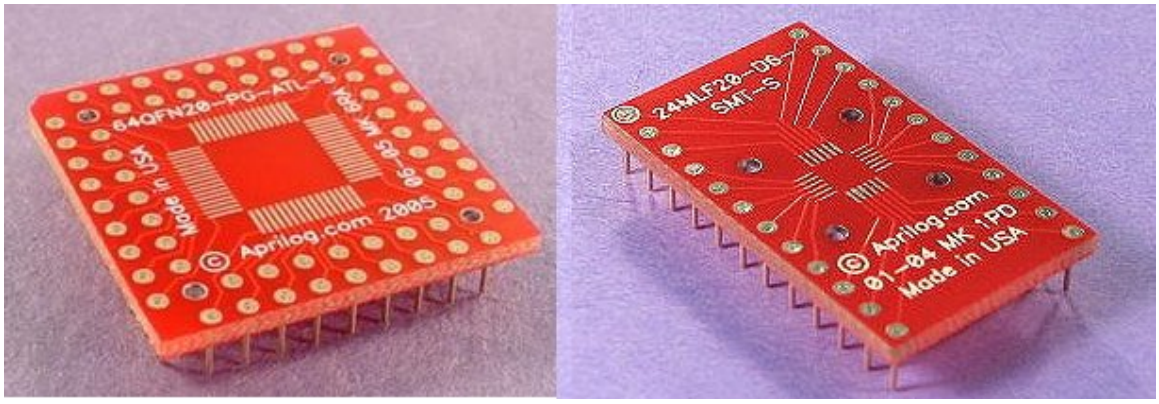


Figure 14. Soldering adapters for QFN packages [29].

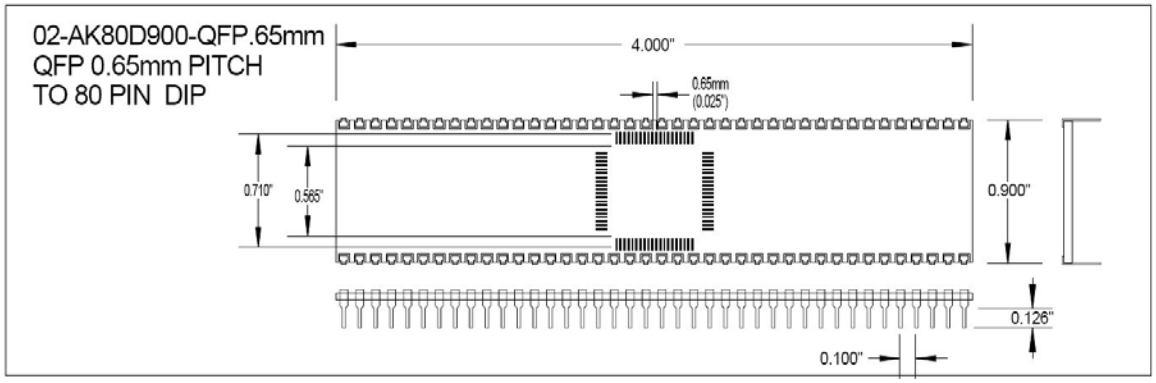


Figure 15. Soldering adapter for QFP package [30].

Prior to ordering the adapters, the plan to solder the discrete devices was to use an optical lab tool that can place such devices and manually apply heat with a heat-gun to melt the solder. However, after receiving both the adapters and the IC samples it became apparent that this plan would not be feasible due to the fine pitch of the packages. Fortunately a local PCB assembly house, VirTex Assembly services, came to the rescue and offered us a low-cost option to attach the ICs to their respective adapters. VirTex

had both the tools and skills to perform such a delicate operation which was far beyond the novice skills of the average graduate student.

6.4 CUSTOM PGA CONNECTION

In the case of the AD6653 receiver, the purchased connector was pinned out as a pin grid array (PGA) instead of the ideal dual in-line package (DIP) configuration. Although an adapter was found that would allow the AD6653 to be soldered directly to a DIP connection, it was not available for purchase. This necessitated yet another level of adaptation. After much searching, it was found that wire-wrap pins would make a solid connection to both the PGA and the female end of a ribbon connector. This allowed for a series of ribbon cables to be attached to the adapter which could easily be connected to a solderless breadboard with a male-to-male connector or a simple 22 gauge wire. The time constraints on the project prevented the custom adapter from being fully tested, but the solution was theoretically sound.

6.5 CSPI DRIVER

One of the design choices made for the project was to choose components that were programmable through a SPI because of the CSPI that was available on the chosen i.MX21 processor. The thought process was that a configurable driver could likely be downloaded for the i.MX21. Secondly, it was also thought that by hooking up the appropriate four-wire interface there would be very little room for error and this would ease the task of debug. This assumption turned out to be false.

A short Internet search yielded a CSPI Linux driver for the i.MX21, but the initial attempts at compilation had numerous errors. After several hours were spent trying to

resolve the non-compiling driver issues, this strategy was abandoned and the Freescale Helpdesk was contacted for support. They were able to provide yet another driver that did compile, but was a bit more complex than what was required for this project. Eventually it was decided a custom-driver was needed, and the example sent to us by Freescale was utilized as a template. The CSPI driver was successfully debugged, and the following scope plots were taken on the TLL6219 daughterboard. Figure 16 shows the SCLK and CS signals, and Figure 17 shows the data signals. Figure 18 is the requirements for the SPI interface from the first target application, the AD9857. Appendix D holds the source code for the driver which was a co-development subproject by Perkey and Canac.

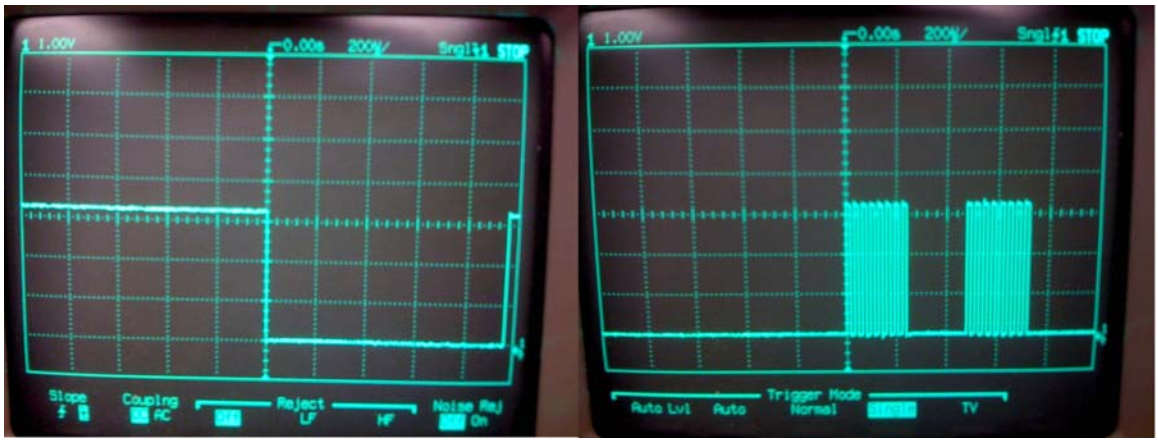


Figure 16. Scope plots of CS and SCLK signals.

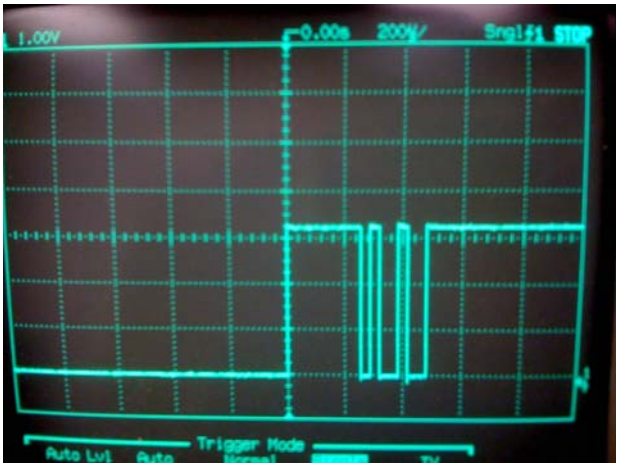


Figure 17. Scope plot of CSPI data.

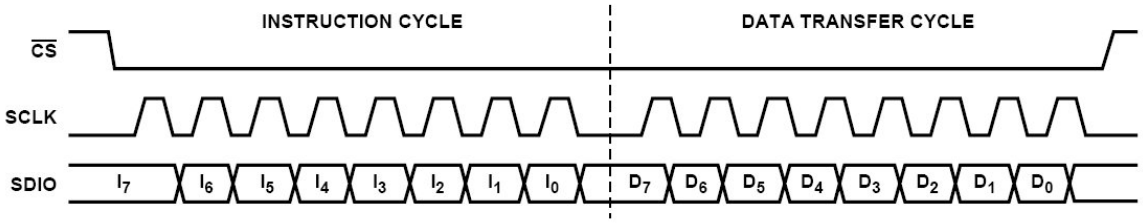


Figure 18. AD9857 SPI Write Timing - Clock Stall Low [21].

6.6 VERILOG TESTING DRIVER

To facilitate debug efforts, digital interface boards were located that would allow access to the TLL5000 mezzanine connectors and thusly into the FPGA. These simple pass-through boards routed the signals to standard 40-pin ribbon cable sockets via buffers that provided voltage translation if required. In addition, this pass-through gave additional electrostatic discharge (ESD) protection to the TLL5000 baseboard.

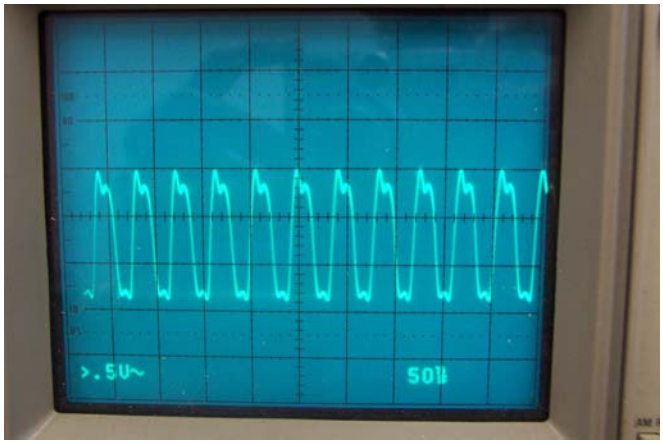


Figure 19. Clock signal generated for debug in the FPGA.

Habib wrote a Verilog driver for the FPGA to control the hardware digital interface boards during debug. This generated both a LVPECL clock from the baseboard software and also a CMOS 3.3V clock generated directly from the FPGA. Although the LVPECL clock was the intended driver for the PCB, the signal as scoped at the solderless PCB appeared too weak to perform useful clocking. As a result, the CMOS 3.3V clock was used for debug as shown in Figure 19. It is understood that this type of clock signal would have too much jitter for the RF components, but it was determined this would suffice for the initial debug of digital baseband components.

PART 3: RESULTS, CONCLUSION, AND FUTURE WORK

Chapter 7: Results

In Chapter 7 the status of the hardware design and the results of the verification and debug processes are discussed. Additionally, the current state of the measurements from the test plan are described. Based on these results, the future work and next steps for the project are outlined as well.

7.1 DEBUGGING EXERCISES

To begin building and debugging the hardware schematics, the TX processor was first mounted to a breadboard and wired up as shown in Figure 20. The goal was to first establish communication with the processor, program the registers, and put it into single-tone mode, and measure different frequency sine waves from the output. This was the logical first step for our debugging efforts, and this was where the hardware verification began as soon as all of the supplies were available.

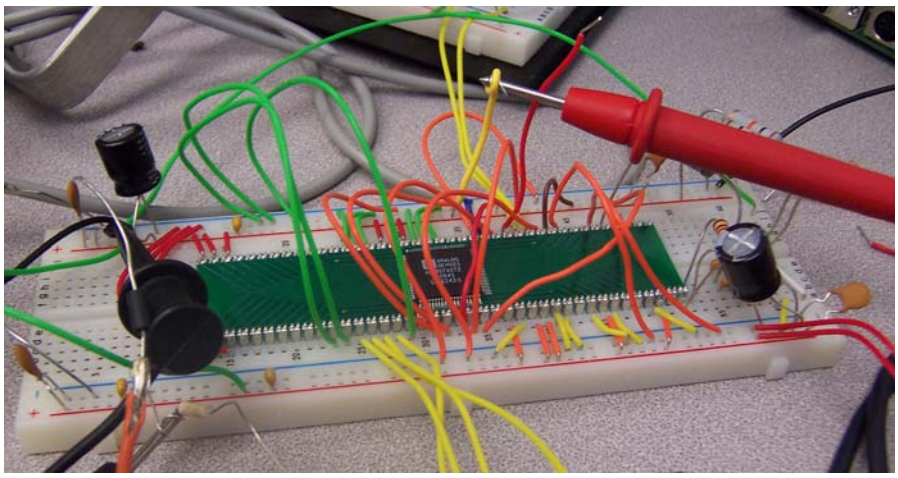


Figure 20. Debug setup of TX processor AD9857.

However communication with the processor, the first step to debugging the system, was not completed within the timeline for the project. Although the CSPI driver appears to be functioning as programmed, the output pin of the AD9857 never came low or changed values. There are multiple conclusions that could be drawn from this. First, potentially the processor is wired incorrectly. A rewire could take place to check for possible issues. If this approach were taken, one potential difference between the first attempt would be to skip the separation of the analog AVDD 3.3-V supply from digital DVDD 3.3-V supply. It was the author's intention to keep these supplies as separate as possible on the breadboard for noise reduction; however this did complicate the hand-wiring scheme.

A second possibility for why the serial output of the AD9857 never changed may have been that the particular IC was damaged during wiring or debug by ESD. Towards the end of the project a replacement of the IC was done with no success in resolving the issues. Although it is possible both units used were damaged, it is considered unlikely due to both the caution used in the lab combined with the ESD protection circuitry that is present in the design of the AD9857. Although the data sheet does not specifically list the conditions to which they were tested, it is a reasonable assumption that normal human-body model (HBM) levels were asserted and passed.

Another feasible reason for the lack of communication could be the boot sequence. There is little documentation on this topic in the datasheet for the AD9857, and the sequence mentioned there was replicated in the lab by hand. In other words, the clocks were applied while reset was held low, then the wire connecting reset was moved to high. The SYNCIO pin was also used to manually reset the SPI port, but again this was done by hand and was potentially very bouncy. This sequence could be automated through further development of the CSPI driver by adding specific GPIO pins for the

boot sequence and control signals. This was the intended path of development for the driver, but development was cut short due to the time constraints of the debug exercises.

Contact with Analog Devices technical support was initiated, but this approach was unfortunately started too late in the project to provide meaningful assistance within the project timeline. Although they were very supportive, the communication lag was too great for the project schedule.

The final lab setup is shown in Figure 21. Starting at the top left and working clockwise is an oscilloscope with the system clock, the TLL5000 baseboard, an ammeter (grey) reading the current-draw of the breadboard, the ATX power supply, and a voltmeter (yellow) measuring the output of the 3.3-V supply. In the center is the AD9857 breadboard from Figure 20.

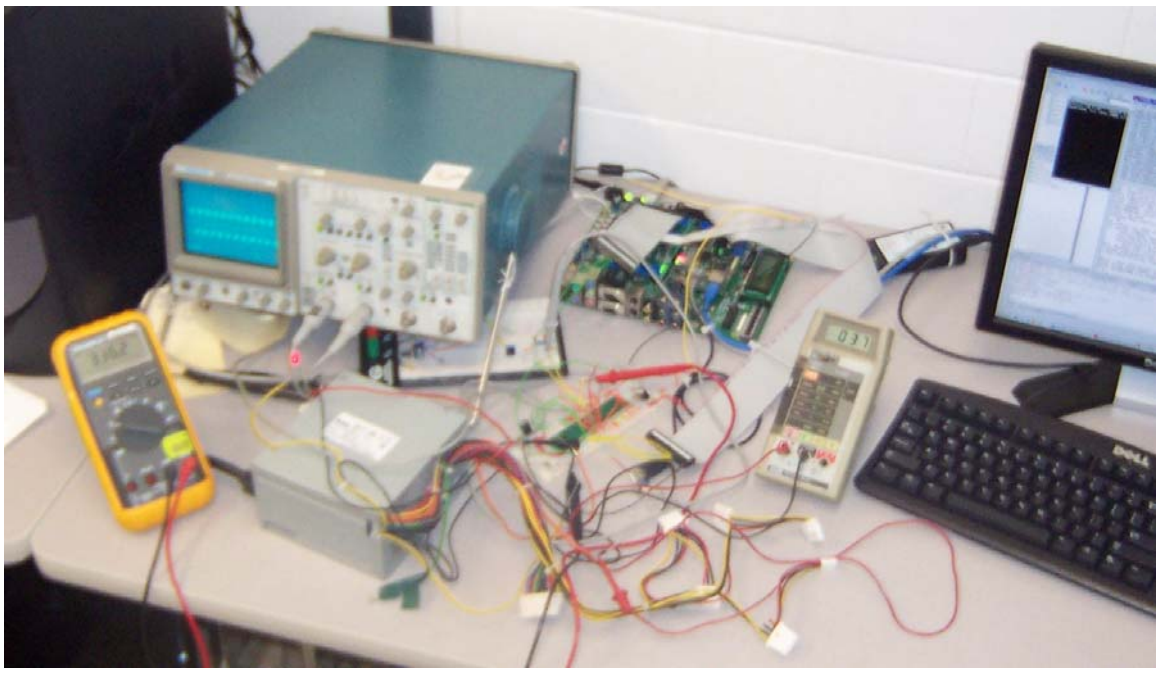


Figure 21. Lab setup for prototype debug.

7.2 TEST PLAN MEASUREMENTS STATUS

Under the test plans that were developed, the ECC and PHY-layer algorithms were setup with the ability to be tested independently. The final status of the hardware was not far enough along to execute any of the planned hardware measurements. This of course also hindered any system-level measurements of the system that were planned. During the research phase of the project, [31] was found and gave way to several additional measurements that would be beneficial after the system is completed. For example, the TX/RX symmetry measurement that was taken would be an excellent measure of the new system. It is possible that some of the design reasoning behind the new PCBs would improve the results seen by Mandke *et al*.

Overall, the project results were largely defined by timeline constraints. Due to the scope plots taken of the CSPI driver, the problematic area preventing programming of the AD9857 is most likely on the breadboard. It is expected that with a few additional weeks of debug, these issues could be resolved and would enable the output of a programmable sine wave in single-tone mode. With this working, a test data sequence could easily be written in Verilog to drive the transmitter, and the output could again be measured with the oscilloscope at the output pin of the AD9857.

Chapter 8: Conclusion and Future Work

Over the course of the project, several key deliverables were met towards the end goal of extending the capabilities of the TLL5000 by implementing a wireless transceiver. At the Data Link Layer, error correction and transmission protocols have been defined, implemented, and tested. In the Physical Layer, algorithms for modulation, demodulation, equalization, and timing recovery have been identified, verified through simulation in MATLAB, and translated into C. For the system hardware, schematics have been generated for the full RF board design, drivers have been written for SPI communication, and prototype verification has begun for the baseband half of the design. The sum of these deliverables, while not completing the entirety of the project, provide an excellent foundation for continuation of the implementation for the Wireless TLL Transceiver.

Simpler routes could have been taken along the way. For instance, there were low-bandwidth single-solution ICs that were discussed during the research portion of the project [32] - [35]. These solutions could be potentially viable options if the goal of educational wireless transmission were chosen over the bandwidth goals. However, the hope was to include sufficient bandwidth for at least audio if not image or low-resolution video transmission. These goals, nevertheless, are incongruent with the ISM band selection necessary for an international audience. The amateur band within the United States gave the freedom to experiment with such performance, but that would be in violation of RF regulations elsewhere in the world.

Another route that would have greatly simplified the project would be to tape-out the original design, a copy of the RF components from the USRP project. Although from a systems design point of view this is academically uninteresting, from a pragmatic

perspective this would be the easiest route towards working hardware. Enhancements to the USRP present in the USRP2, released in May of 2009, can be referenced for potential improvements [11]. The difficulty with this option is that the noise sources that were warned against in Section 5.1 may plague the initial design, and possibly require a second or third re-spin of the board. The USRP reference design itself went through several revisions, and it is probable that the RFX designs also needed multiple revisions as well.

As of November 2009, the AD6653 has been discontinued by Analog Devices according to their website (www.analog.com). In the design phase, the author attempted to avoid the possible issue of discontinued ICs by using more recently released products. The AD6653, for instance, had a datasheet published in 2007; making it a particularly young product to be discontinued. In future work, selecting a different IC may offer a way to design around this unforeseen complication.

From a systems design perspective, there is much work that needs to be done. The ECC and PHY-level algorithms need to be profiled on the ARM9 processor to see how much processing is already consumed by the existing code, how much processing power is left for higher-level user interface code, and what functions would be the best candidates for acceleration within the FPGA. The breadboard prototype needs to be completed and debugged for the baseband planes before considering what strategy would be best to validate the RF-plane design. Finally, a layout and fabrication of the full PCB needs to be completed and verified with the testbench established on the prototype hardware. With these tasks completed, the original test plan can be executed to properly gauge the metrics of the design.

Even in light of the described potential alternatives, the best approach to complete this project would be to continue with the present work. It is the most difficult path but has the most potential for an ideal solution. The Wireless TLL Transceiver with the

complete software and Verilog solution working as a single package would present a powerful learning and development tool in the area of embedded systems design for wireless communication. The addition of this to a curriculum would only help solidify theoretical concepts studied in RF design, communication theory, and embedded systems design. Furthermore, it would add an excellent research tool for students to develop PHY-level designs and evaluate new communication algorithms. These new designs, combined with the ARM architecture, could model communication systems intended for embedded systems.

Appendices

APPENDIX A: ACRONYM DEFINITIONS

ADC – Analog to Digital Converter

ARM – Advanced RISC Machine

ASIC – Application Specific Integrated Circuit

ATX – Advanced Technology Extended

BER – Bit Error Rate

BPSK – Binary Phase-Shift Keying

CIC – Cascaded Integrator-Comb

CMOS – Complementary Metal-Oxide-Semiconductor

CSPI - Configurable Serial Port Interface

DAC – Digital to Analog Converter

DDC – Digital Downconverter

DDS – Direct Digital Synthesis

DIP – Dual In-line Package

DSP – Digital Signal Processor

DUC – Digital Upconverter

ECC – Error-Correction Code

EDA – Electronic Design Automation

EEPROM – Electronically Erasable Programmable Read-Only Memory

ESD – Electrostatic Discharge

FCC – Federal Communications Commission

FIR – Finite Impulse Response

FPGA – Field-Programmable Gate Array

GPP – General Purpose Processor

GPS – Global Positioning System

HBM – Human-Body Model

HDL – Hardware Description Language

I and Q – In-phase and Quadrature

I^2C – Inter-Integrated Circuit

I/O – Input / Output

IC – Integrated Circuit

IF – Intermediate Frequency

ISM – Industrial, Scientific, and Medical

LPF – Low Pass Filter

LVDS – Low-Voltage Differential-Signaling

LVPECL – Low-Voltage Positive Emitter-Coupled Logic

MSPS – Mega Samples Per Second

MxFE – Mixed-Signal Front End Processor; MxFE is a trademark of Analog Devices, Inc.

NCO - Numerically-Controlled Oscillator

OSI – Open Systems Interconnection

PA – Power Amplifier

PC – Personal Computer

PCB – Printed Circuit Board

PDF – Portable Document Format

PGA – Pin Grid Array

PGA – Programmable-Gain Amplifier

PHY – Physical Layer

QFN – Quad Flat No Leads

QFP – Quad Flat Package

QPSK – Quadrature Phase-Shift Keying

RF – Radio-Frequency

RFX – Radio-Frequency Transmitter

RISC – Reduced Instruction Set Computer

RX – Receiver

SHA – Sample and Hold Amplifier

SMA – SubMiniature version A

SMSC – Smart Mixed-Signal Connectivity

SPI – Serial Port Interface

TLL – The Learning Labs

TX – Transmitter

uMON – MicroMonitor boot loader

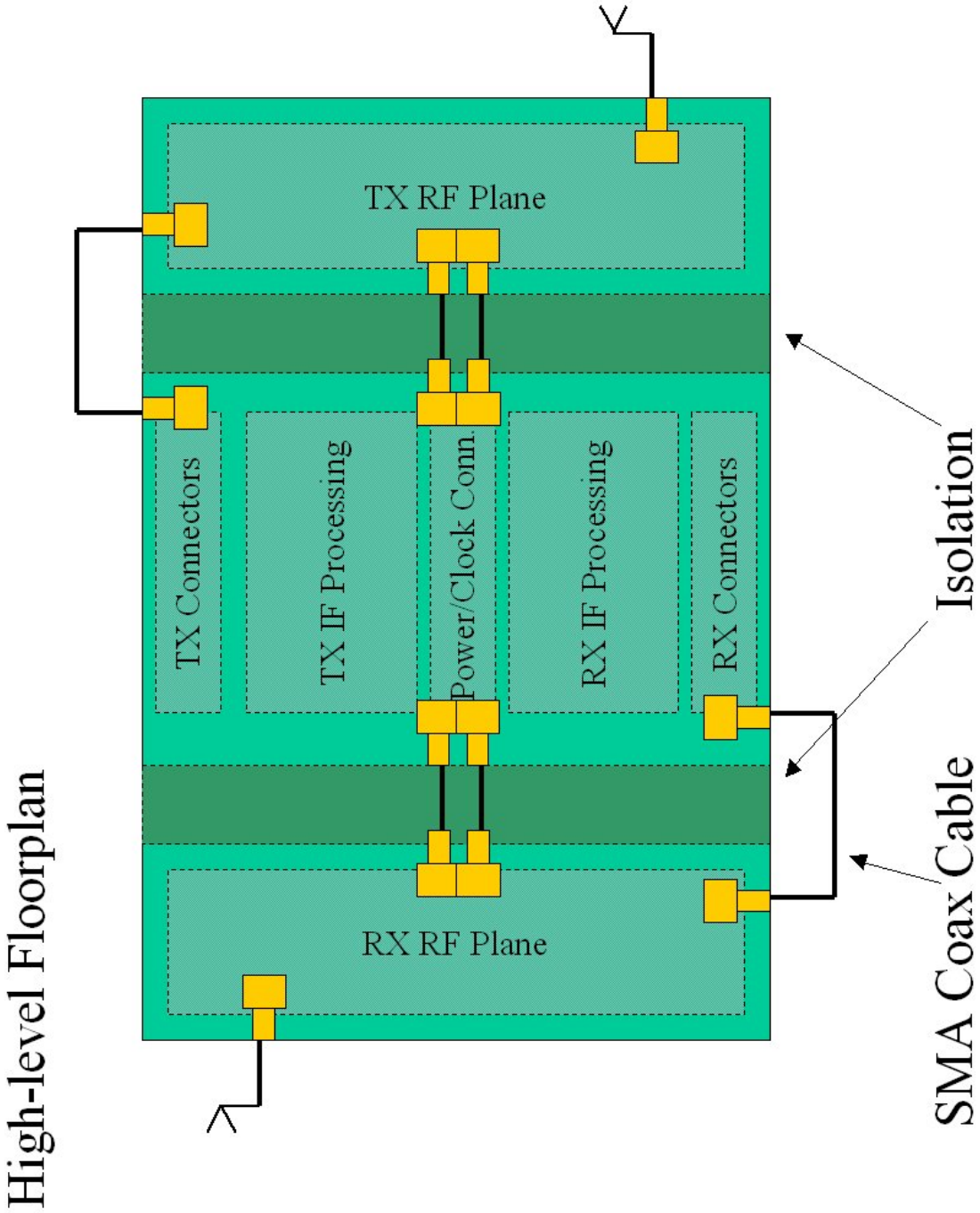
USB – Universal Serial Bus

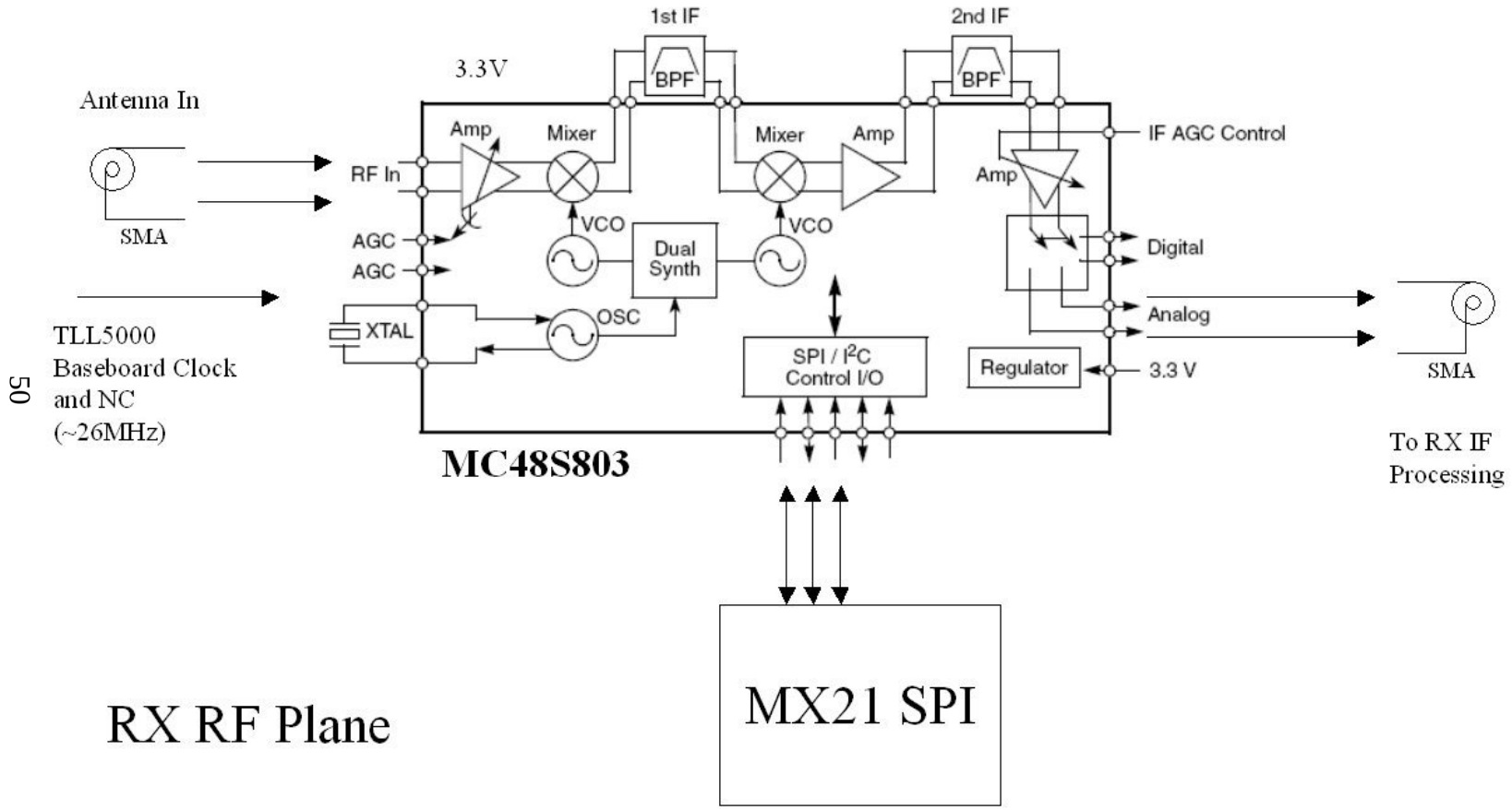
USRP - Universal Software Radio Peripheral (pronounced “usurp”)

UT – The University of Texas at Austin

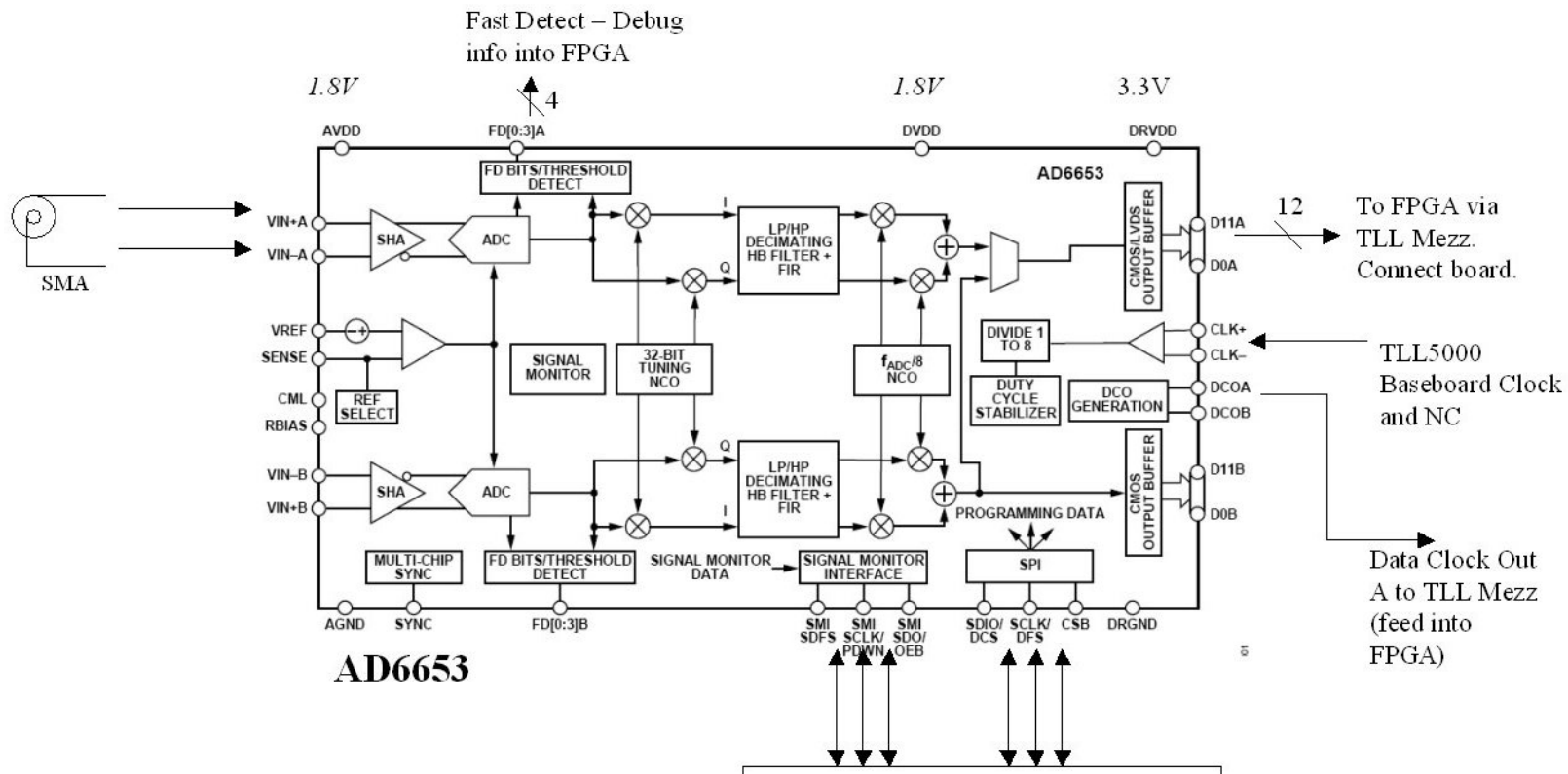
VGA – Variable-Gain Amplifier

APPENDIX B: FLOORPLAN AND SYSTEM BLOCK DIAGRAM




1st IF
2nd IF
3.3V
BPF
BPF
Antenna In
Amp
Mixer
Mixer
Amp
IF AGC Control
Amp
RF In
SMA
VCO
VCO
Dual
Synth
Digital
AGC
AGC
Analog
XTAL
OSC
SPI / I2C
Control I/O
Regulator
3.3 V
SMA
TLL5000
Baseboard Clock
and NC
(~26MHz)
MC48S803
To RX IF
Processing
MX21 SPI

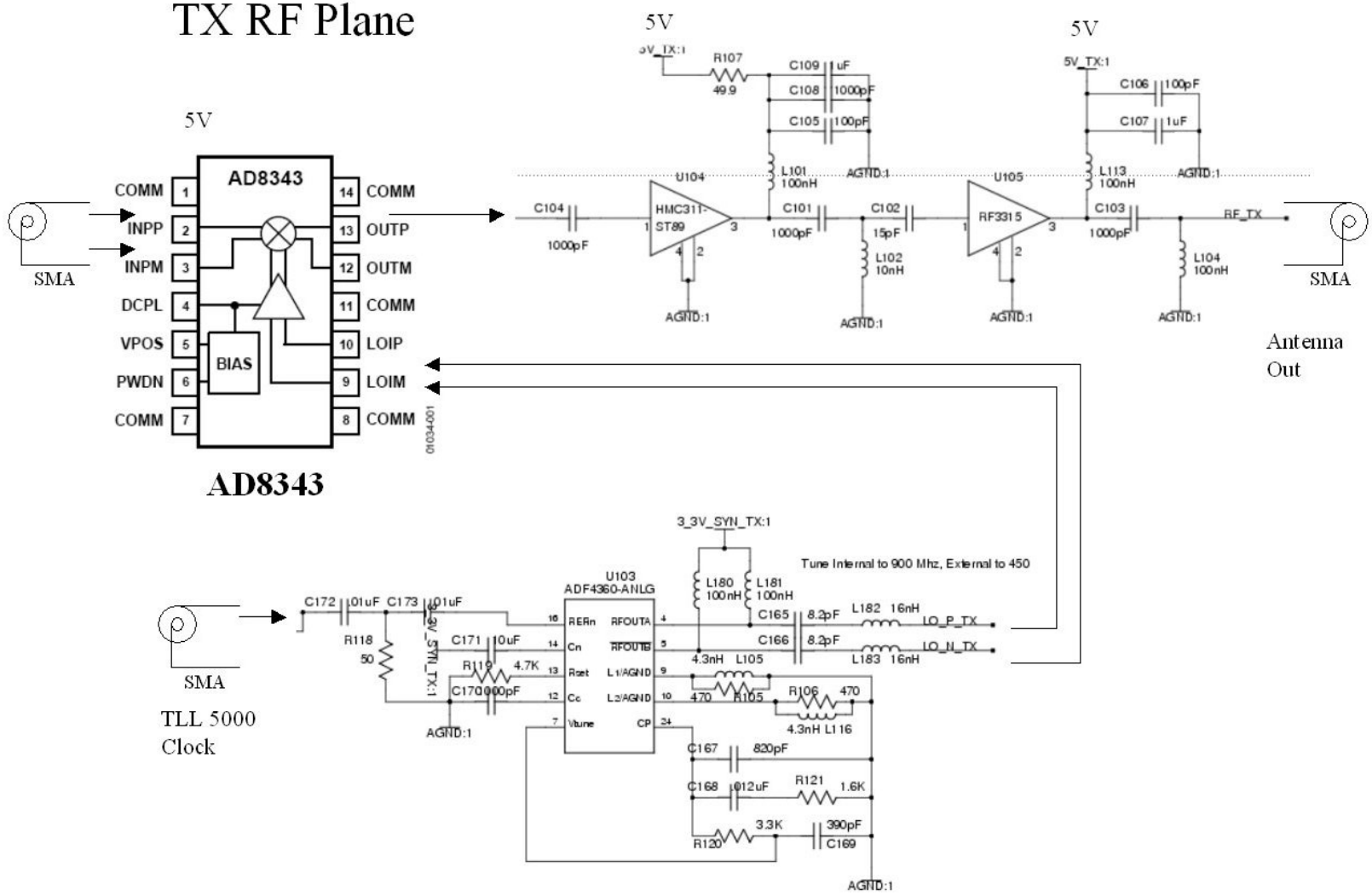
RX RF Plane

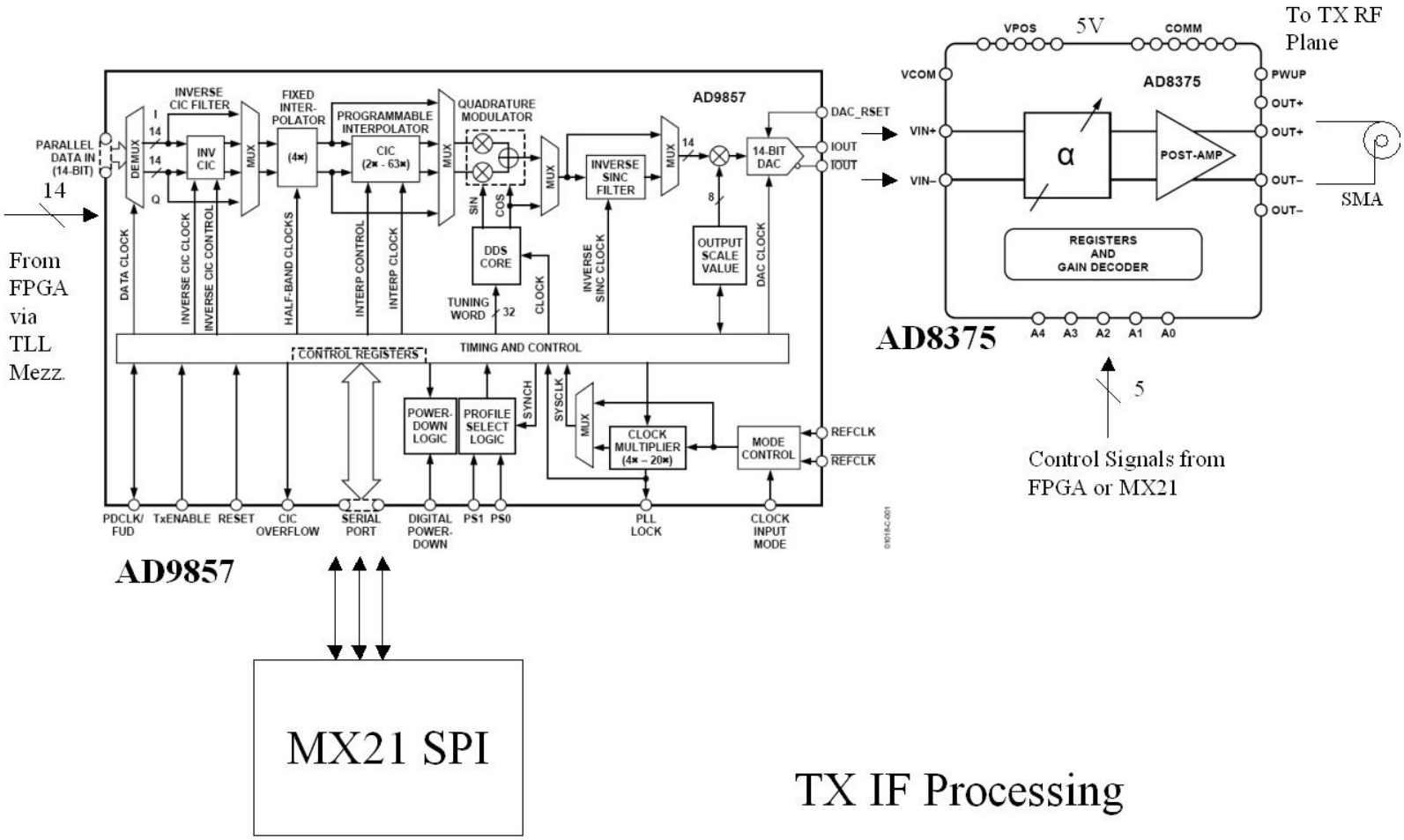


RX IF Processing

MX21 SPI

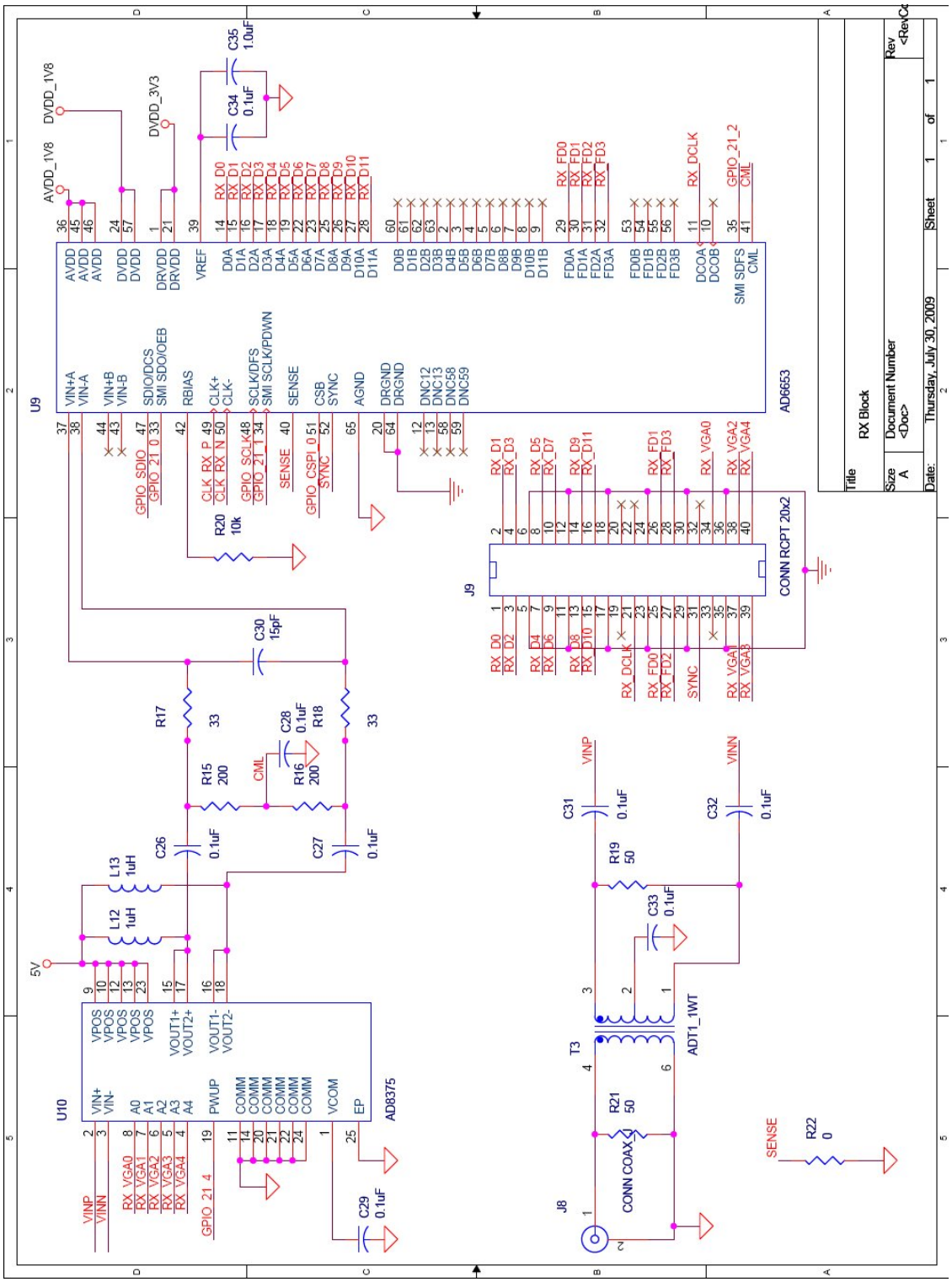
TX RF Plane

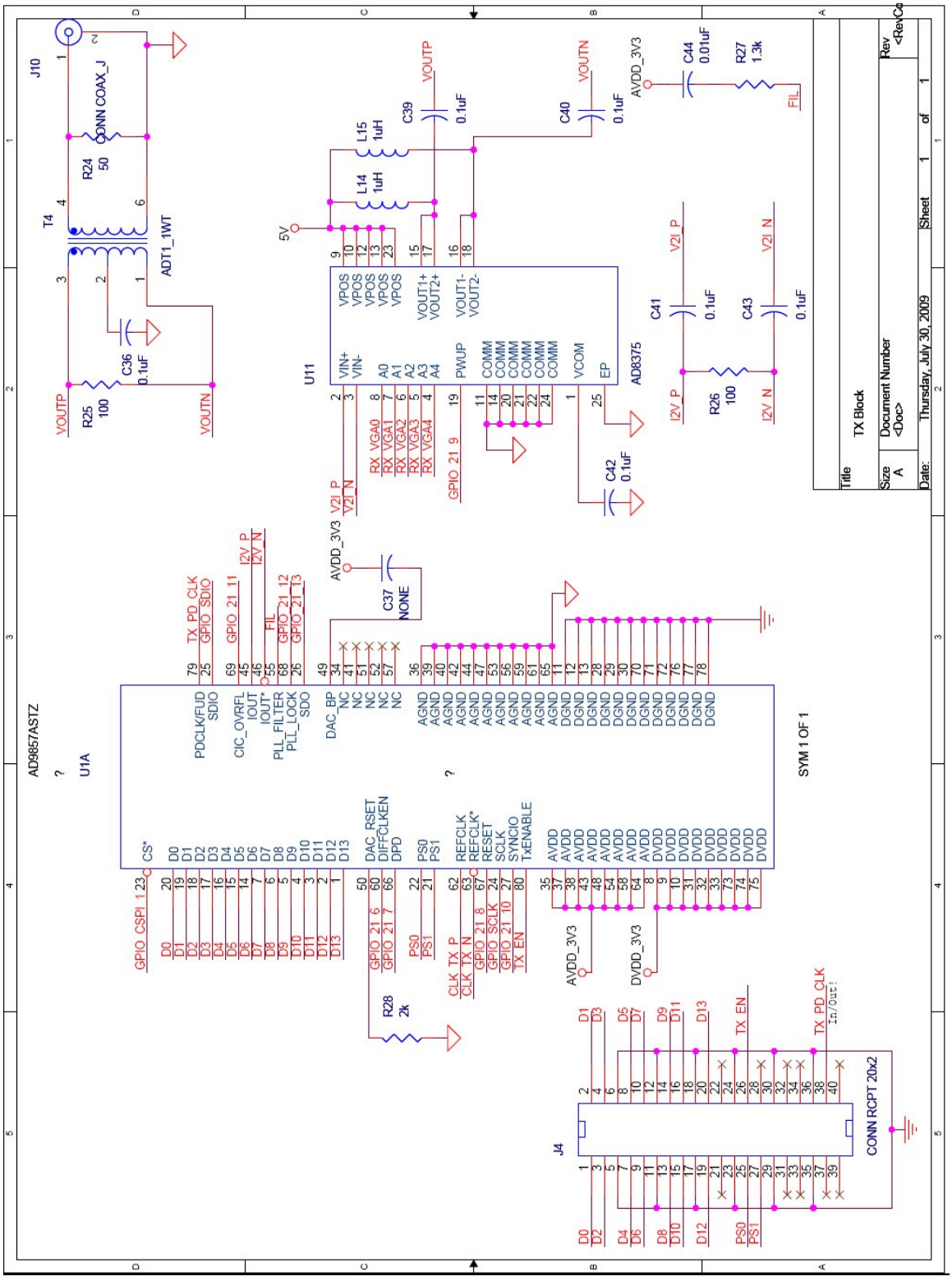

5V
5V
5V
AD8343
COMM
INPP
INPM
DCPL
VPOS
PWDN
COMM
BIAS
OUTP
OUTM
LOIP
LOIM
SMA
SMA
Antenna Out
U104
HMC311-ST89
U105
RF3315
R107
49.9
C109 uF
C108 1000pF
C105 100pF
C106 100pF
C107 1uF
L101 100nH
L113 100nH
C104 1000pF
C101 1000pF
C102 15pF
C103 1000pF
L102 10nH
L104 100nH
RF_TX
AGND:1
AD8343
3_3V_SYN_TX:1
Tune Internal to 900 Mhz, External to 450
U103
ADF4360-ANLG
L180 100nH
L181 100nH
C165 8.2pF
C166 8.2pF
L182 16nH
L183 16nH
LO_P_TX
LO_N_TX
4.3nH L105
R106 470
4.3nH L116
C172 .01uF
C173 .01uF
R118 50
C171 10uF
R119 4.7K
C170 1000pF
C167 820pF
C168 .012uF
R121 1.6K
R120 3.3K
C169 390pF
SMA
TLL 5000 Clock

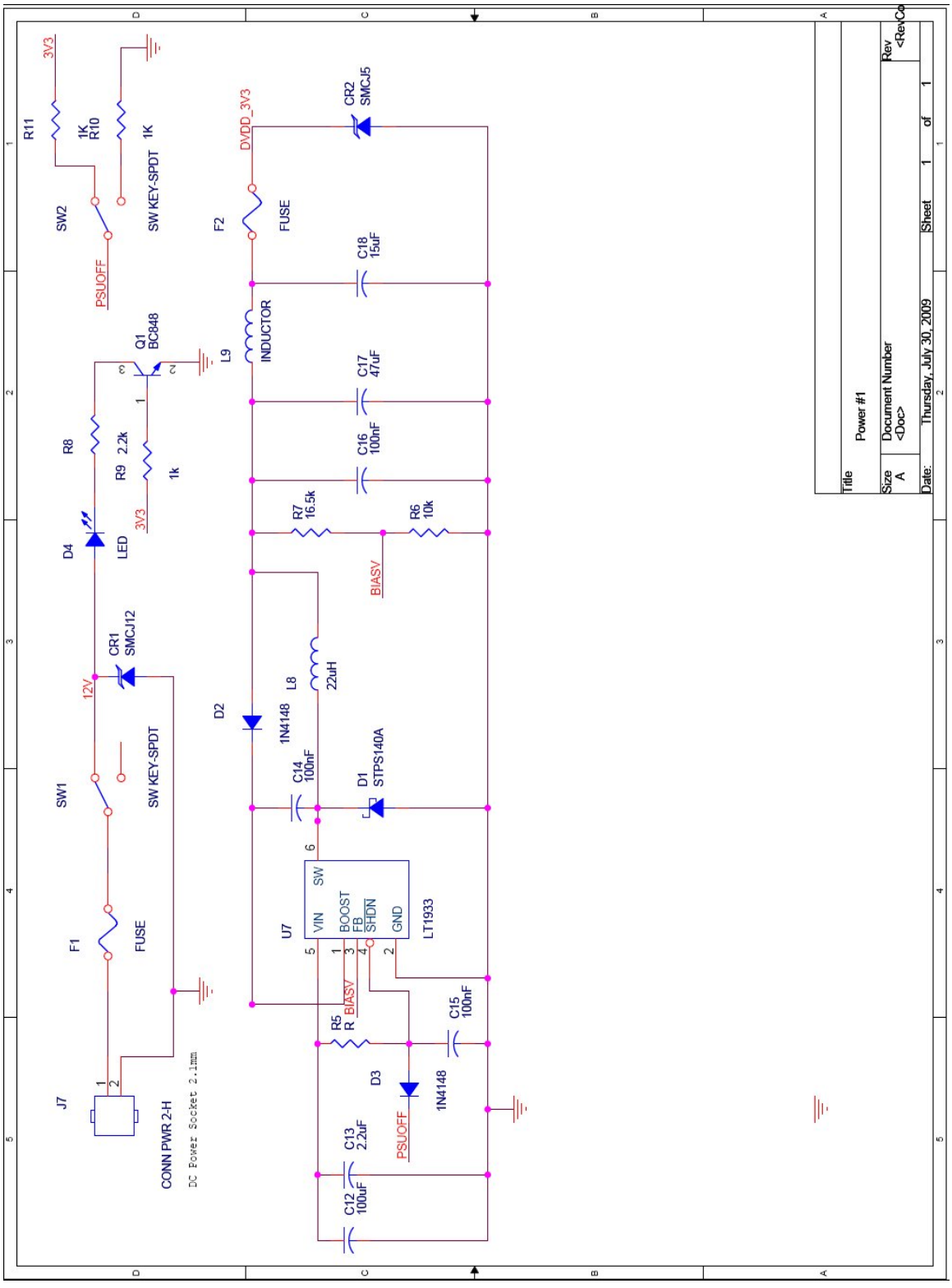


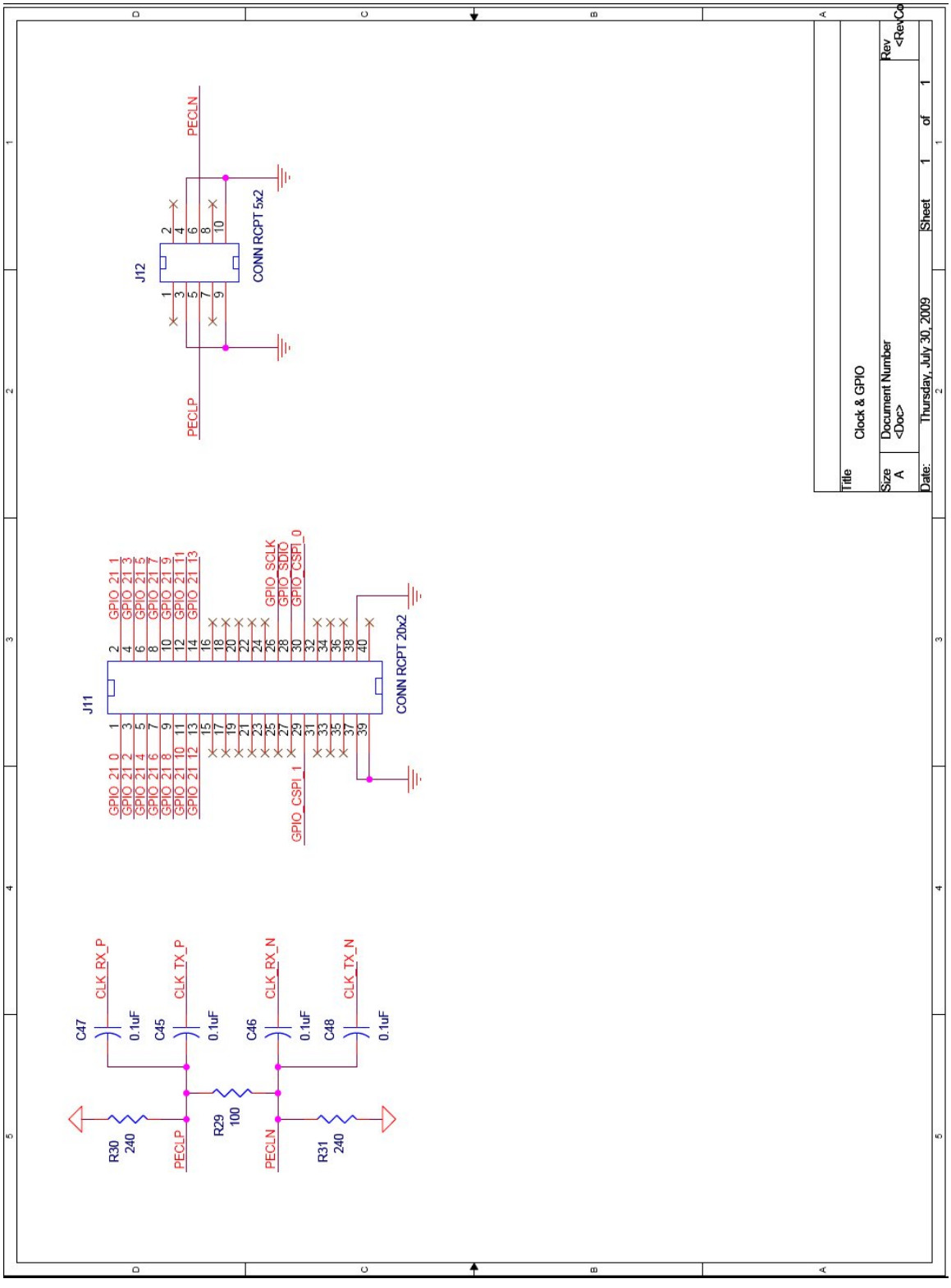
TX IF Processing

APPENDIX C: DESIGN SCHEMATICS




AD9857/ASTZ
U1A
SYM 1 OF 1
J10
CONN COAX_J
T4
ADT1_1WT
R24 50
R25 100
C36 0.1uF
VOUTP
VOUTN
U11
AD8375
L14 1uH
L15 1uH
C39 0.1uF
C40 0.1uF
C41 0.1uF
C42 0.1uF
C43 0.1uF
C44 0.01uF
R26 100
R27 1.3k
C37 NONE
R28 2k
J4
CONN RCPT 20x2
AVDD_3V3
DVDD_3V3
5V
I2V_P
I2V_N
V2I_P
V2I_N
TX_EN
TX_PD_CLK
FIL
Title
TX Block
Size A
Document Number
Rev
Date: Thursday, July 30, 2009
Sheet 1 of 1


3V3
R11
1K
R10
1K
SW2
SW KEY-SPDT
PSUOFF
Q1
BC848
R8
R9 2.2k
1k
3V3
D4
LED
12V
CR1
SMCJ12
SW1
SW KEY-SPDT
F1
FUSE
J7
CONN PWR 2-H
DC Power Socket 2.1mm
DVDD_3V3
F2
FUSE
L9
INDUCTOR
CR2
SMCJ5
C18
15uF
C17
47uF
C16
100nF
R7
16.5k
R6
10k
BIASV
D2
1N4148
L8
22uH
C14
100nF
D1
STPS140A
U7
VIN
SW
BOOST
FB
SHDN
GND
LT1933
R5
R
BIASV
C15
100nF
D3
1N4148
PSUOFF
C13
2.2uF
C12
100uF
Title
Power #1
Size
A
Document Number
<Doc>
Rev
<Rev
Date:
Thursday, July 30, 2009
Sheet
1
of
1


J12
CONN RCPT 5x2
PECLN
PECLP
J11
CONN RCPT 20x2
GPIO_21_0
GPIO_21_1
GPIO_21_2
GPIO_21_3
GPIO_21_4
GPIO_21_5
GPIO_21_6
GPIO_21_7
GPIO_21_8
GPIO_21_9
GPIO_21_10
GPIO_21_11
GPIO_21_12
GPIO_21_13
GPIO_SCLK
GPIO_SDIO
GPIO_CSPI_0
GPIO_CSPI_1
C47
C45
C46
C48
0.1uF
CLK_RX_P
CLK_TX_P
CLK_RX_N
CLK_TX_N
R30
240
R29
100
R31
240
PECLP
PECLN
Title
Clock & GPIO
Size
A
Document Number
<Doc>
Rev
<RevCo
Date:
Thursday, July 30, 2009
Sheet
1
of
1

APPENDIX D: CSPI DRIVER SOURCE CODE

```
/*
 *  mycspi.c
 */

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/version.h>
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <linux/interrupt.h>
#include <linux/sched.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/vmalloc.h>
#include <linux/mman.h>
#include <linux/slab.h>
#include <linux/ioport.h>
#include "mycspi_memmap.h"

#if LINUX_VERSION_CODE < KERNEL_VERSION(2,6,0)
#include <linux/wrapper.h>
#endif

#define mx21_VERSION "1.0"
#define mx21_MAJOR 240
#define mx21_NAME "clocks"

#define SPI1 1
#define SPI2 2
#define SPI3 3
#define DIV_BY_32 0x08
#define DIV_BY_1024 0x12
#define DIV_BY_512 0x11
#define SLAVE_ 0
#define MASTER_ 1
#define SS_ACTIVE_LOW 0
#define PHA0 1
#define POL_ACTIVE_LOW 1
#define BITCOUNT_16 15
#define BIT_COUNT_8  7

#define COMMAND_MASK 0xf0000000
#define CHG_CLK_F1_CMD 0x80000000
#define AD9857_F2_CMD 0x40000000
#undef DEBUG

//volatile unsigned int *pllclk_pccr0;
volatile PLLCLK_PCCR0reg   * pllclk_pccr0;
volatile PLLCLK_PCCR1reg   * pllclk_pccr1;
volatile PLLCLK_PCDR1reg   * pllclk_pcdr1;
volatile GPIO_PTD_GIUSreg  * gpio_ptd_gius;
volatile GPIO_PTE_GIUSreg  * gpio_pte_gius;
volatile GPIO_PTE_GPRreg   * gpio_pte_gpr;
volatile GPIO_PTE_DDIRreg  * gpio_pte_ddir;
volatile GPIO_PTE_DRreg    * gpio_pte_dr;
volatile GPIO_PTE_OCR2reg  * gpio_pte_ocr2;
volatile CSPI2_RESETreg    * cspi2_reset;
volatile CSPI2_CONTROLreg  * cspi2_control;
volatile CSPI2_TXDATAreg   * cspi2_txdata;
volatile CSPI2_TESTreg     * cspi2_test;
```

```
volatile CSPI2_SAMP_PERreg * cspi2_samp_per;
volatile CSPI2_RXDATAreg   * cspi2_rxdata;

static int mx21_open1 (struct inode *inode, struct file *file) {
   return 0;
}

static int mx21_release1 (struct inode *inode, struct file *file) {
   return 0;
}

static int mx21_ioctl1(struct inode *inode, struct file *file, unsigned int cmd, unsigned
long arg) {

   int retval = 0;
   int input;
   unsigned long value;
   unsigned int command_type;
   unsigned int offset;
   int datarate;
   int mode;
   int sspol;
   int pha;
   int pol;
   int bitcount;
   int failCount = 0;
   int test;
   unsigned int my_cspi2_rx_data;
   unsigned int my_cspi2_rx_fifo[10];
   int i, y;

   //command_type = 0x80000000ul;

   // Set the offset for register accesses
   command_type = COMMAND_MASK & cmd;

   switch(command_type)
   {
      case 0:
       // enable PERCLK to CSPI2 and GPIO module
       pllclk_pccr0->bits.GPIO_EN = 1;
       pllclk_pccr0->bits.CSPI1_EN = 0;
       pllclk_pccr0->bits.CSPI2_EN = 1;
       pllclk_pccr1->bits.CSPI3_EN = 0;

       // set up PERCLK2 divide, divide MCU PLL by 4 (66MHz PERCLK2, assuming
MCUPLL=266MHz)
       //pllclk_pcdr1->bits.PERDIV2 = 4;

       // enable CSPI functionality from GPIO
       gpio_ptd_gius->bits.PIN31 = 0;
       gpio_ptd_gius->bits.PIN30 = 0;
       gpio_ptd_gius->bits.PIN29 = 0;
       gpio_ptd_gius->bits.PIN28 = 0;
       gpio_ptd_gius->bits.PIN27 = 0;
       gpio_ptd_gius->bits.PIN26 = 0;
       gpio_ptd_gius->bits.PIN25 = 0;
       gpio_ptd_gius->bits.PIN24 = 0;
       gpio_ptd_gius->bits.PIN23 = 0;
       gpio_ptd_gius->bits.PIN22 = 0;
       gpio_ptd_gius->bits.PIN21 = 0;
       gpio_ptd_gius->bits.PIN20 = 0;
       gpio_ptd_gius->bits.PIN19 = 0;

       //configure gpio pin PTE23 (shared with CSPI3_SCLK
       //to initialize state of ad9857
```

```
 gpio_pte_gius->bits.PIN23 = 1; //init to use GPIO rather than muxed functions
 gpio_pte_ddir->bits.PIN23 = 1; //init to use GPIO rather than muxed functions
 gpio_pte_ocr2->bits.PIN23 = 0x3; //indicate get data from DR register
 gpio_pte_dr->bits.PIN23   = 0;
 //wait(1);
 for(y; y<5000; y++){ }
 gpio_pte_dr->bits.PIN23   = 1;
 //wait(1);
 for(y; y<500; y++){ }
 gpio_pte_dr->bits.PIN23   = 0;

 // reset CSPI2
 cspi2_reset->bits.START = 1;

 // set up data rate
 //datarate = DIV_BY_1024;       // divide by 12
 datarate = DIV_BY_512;       // divide by 11
 cspi2_control->bits.DATA_RATE = datarate;

 // set the SPI mode as master(1) or slave(0)
 mode      = MASTER_;         // use SPI2 as master
 cspi2_control->bits.MODE = mode;

 // set up the Slave Select polarity, 0=ActiveLow, 1=ActiveHigh
 sspol     = SS_ACTIVE_LOW;   // slave select is active when low
 cspi2_control->bits.SSPOL = sspol;

 // set up the SPI clock for phase and polarity
 pha       = PHA0;            // phase zero
 cspi2_control->bits.PHA = pha;
 pol       = POL_ACTIVE_LOW;  // polarity zero
 cspi2_control->bits.POL = pol;

 // set bit count
 bitcount  = BIT_COUNT_8;     // 8 bits to transfer each time
 cspi2_control->bits.BIT_COUNT = bitcount;
 // set CS = SS0
 cspi2_control->bits.CS = 0;

 //set Sample Period
// cspi2_samp_per->bits.WAIT = 0x7FFF;
 cspi2_samp_per->bits.WAIT = 0x02;

 // enable_SPI2
 cspi2_control->bits.SPIEN = 1;

break;

case CHG_CLK_F1_CMD:
  // put data into the FIFO
  for (value=0; value <10; value++){
    printk("\nValue %i\n", value);
    cspi2_txdata->all = 0x00001111;
    cspi2_txdata->all = 0x00005555;
    cspi2_txdata->all = 0x0000AAAA;
   // cspi2_txdata->all = 0x00008888;
    test = cspi2_test->bits.TXCNT;
    printk("TXCNT = %d\n", test);
    if (cspi2_test->bits.TXCNT > 0)
    {
      printk("\n**TXCNT TEST FAILED.** TXCNT =%d\n", cspi2_test->bits.TXCNT);
      failCount++;
    }
    // initiate the data exchange
    cspi2_control->bits.XCH = 1 ;
```

```
          while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
        }
        printk("Done with XCH\n");
      break;

       case AD9857_F2_CMD:
        //************************************
        //   AD9857 Instruction Byte Format
        //   MSB                       LSB
        //   D7  D6  D5  D4  D3  D2  D1  D0
        //   RW  N1  N0  A4  A3  A2  A1  A0
        //
        //   RW:  0 = Write, 1 = Read
        //    N:  00 = 1 byte, 01 = 2 bytes, 10 = 3 bytes, 11 = 4 bytes
        //    A:  Register address 0x00 to 0x19
        //************************************
        i=0;
        // put instruction byte into the FIFO
          printk("\nSetting AD9857 to 4-Pin Mode!\n");
          cspi2_txdata->all = 0x0;  // Write 1-byte to address 0x00
        // initiate the data exchange
          cspi2_control->bits.XCH = 1 ;
          while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
          my_cspi2_rx_fifo[i++] = cspi2_rxdata->all;  // Read register contents
        // put data byte into the FIFO
          cspi2_txdata->all = 0xA1;  // Write SDIO Input-Only = 1
          //cspi2_txdata->all = 0xA4;  // Write SDIO Input-Only = 1, PLL = 4x
          cspi2_control->bits.XCH = 1 ;
          while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
          my_cspi2_rx_fifo[i++] = cspi2_rxdata->all;  // Read register contents
          //printk("Done with XCH\n");

          //printk("\nNow read the first byte from AD9857\n");
        // put instruction byte into the FIFO
          cspi2_txdata->all = 0x80;  // Read 1-byte to address 0x00
        // initiate the data exchange
          cspi2_control->bits.XCH = 1 ;
          while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
          my_cspi2_rx_fifo[i++] = cspi2_rxdata->all;  // Read register contents
          cspi2_txdata->all = 0x0;  // Write a dummy data so we can do a real read
        // initiate the data exchange
          cspi2_control->bits.XCH = 1 ;
          while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
          my_cspi2_rx_fifo[i++] = cspi2_rxdata->all;  // Read REAL register contents
          my_cspi2_rx_data = my_cspi2_rx_fifo[i-1];  // Fix the counter index
        // receive a byte into the FIFO
          //cspi2_control->bits.XCH = 1 ;
          //while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
          //my_cspi2_rx_fifo[i++] = cspi2_rxdata->all;  // Read register contents
          //my_cspi2_rx_data = cspi2_rxdata->all;  // Read register contents
          my_cspi2_rx_fifo[i++] = my_cspi2_rx_data;  // Read register contents
          printk("Done with XCH\n");

/*          printk("\nNow read the first byte from AD9857\n");
        // put instruction byte into the FIFO
          cspi2_txdata->all = 0x87;  // Read 1-byte to address 0x00
        // initiate the data exchange
          cspi2_control->bits.XCH = 1 ;
          while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
```

```
          my_cspi2_rx_fifo[i++] = cspi2_rxdata->all;  // Read register contents
          my_cspi2_rx_data = my_cspi2_rx_fifo[i-1];  // Read register contents
        // receive a byte into the FIFO
          cspi2_control->bits.XCH = 1 ;
          while (cspi2_control->bits.XCH == 1); // wait until XCH bit is clear, hence
exchange is complete
          my_cspi2_rx_fifo[i++] = cspi2_rxdata->all;  // Read register contents
          //my_cspi2_rx_data = cspi2_rxdata->all;  // Read register contents
          //my_cspi2_rx_fifo[i++] = my_cspi2_rx_data;  // Read register contents
          printk("Done with XCH\n");
*/

        retval = my_cspi2_rx_data;
        printk("\nData from Register 0x00: %x\n", my_cspi2_rx_data);
          for(i--;i>=0;i--){
                printk("FIFO %d is %x\n", i, my_cspi2_rx_fifo[i]);
          }
      break;

     default:
         retval = -EINVAL;
   }

   return retval;
}

// define which file operations are supported
struct file_operations mx21_fops = {
   .owner=   THIS_MODULE,
   .llseek  = NULL,
   .read    = NULL,
   .write   = NULL,
   .readdir = NULL,
   .poll    = NULL,
   .ioctl   = mx21_ioctl1,
   .mmap    = NULL,
   .open    = mx21_open1,
   .flush   = NULL,
   .release = mx21_release1,
   .fsync   = NULL,
   .fasync  = NULL,
   .lock    = NULL,
   .readv   = NULL,
   .writev  = NULL,
};

// initialize module
static int __init mx21_init_module (void) {
   printk("i.MX21 CSPI2 driver\n");
   printk(KERN_INFO "\nmx21: i.MX21 Driver Loading.\n");
   printk(KERN_INFO "mx21: Using Major Number %d on %s\n", mx21_MAJOR, mx21_NAME);

   if (register_chrdev(mx21_MAJOR, mx21_NAME, &mx21_fops))
   {
      printk("1 mx21: unable to get major %d. ABORTING!\n", mx21_MAJOR);
      return -EBUSY;
   }

//volatile unsigned int *pllclk_pccr0;
   if (check_mem_region(PLLCLK_PCCR0_addr, 0x4))
   {
      printk(KERN_ERR "2 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(PLLCLK_PCCR0_addr, 0x4, mx21_NAME);
```

```
   pllclk_pccr0 = (volatile PLLCLK_PCCR0reg *) __ioremap((unsigned int)
PLLCLK_PCCR0_addr, 0x4, 0);

   if (!pllclk_pccr0)
   {
      printk(KERN_ERR "3 mx21: Unable to map 1.\n");
      release_mem_region((unsigned int)pllclk_pccr0, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *pllclk_pccr1;
   if (check_mem_region(PLLCLK_PCCR1_addr, 0x4))
   {
      printk(KERN_ERR "4 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(PLLCLK_PCCR1_addr, 0x4, mx21_NAME);
   pllclk_pccr1 = (volatile PLLCLK_PCCR1reg *) __ioremap((unsigned int)
PLLCLK_PCCR1_addr, 0x4, 0);

   if (!pllclk_pccr1)
   {
      printk(KERN_ERR "5 mx21: Unable to map 2.\n");
      release_mem_region((unsigned int)pllclk_pccr1, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *pllclk_pcdr1;
   if (check_mem_region(PLLCLK_PCDR1_addr, 0x4))
   {
      printk(KERN_ERR "6 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(PLLCLK_PCDR1_addr, 0x4, mx21_NAME);
   pllclk_pcdr1 = (volatile PLLCLK_PCDR1reg *) __ioremap(PLLCLK_PCDR1_addr, 0x4, 0);

   if (!pllclk_pcdr1)
   {
      printk(KERN_ERR "7 mx21: Unable to map 3.\n");
      release_mem_region((unsigned int)pllclk_pcdr1, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *gpio_ptd_gius;
   if (check_mem_region(GPIO_PTD_GIUS_addr, 0x4))
   {
      printk(KERN_ERR "8 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(GPIO_PTD_GIUS_addr, 0x4, mx21_NAME);
   gpio_ptd_gius = (volatile GPIO_PTD_GIUSreg *) __ioremap(GPIO_PTD_GIUS_addr, 0x4, 0);

   if (!gpio_ptd_gius)
   {
      printk(KERN_ERR "9 mx21: Unable to map 4.\n");
      release_mem_region((unsigned int)gpio_ptd_gius, 0x4);

      return -EBUSY;
   }
```

```
//volatile unsigned int *gpio_pte_gius;
   if (check_mem_region(GPIO_PTE_GIUS_addr, 0x4))
   {
      printk(KERN_ERR "10 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(GPIO_PTE_GIUS_addr, 0x4, mx21_NAME);
   gpio_pte_gius = (volatile GPIO_PTE_GIUSreg *) __ioremap(GPIO_PTE_GIUS_addr, 0x4, 0);

   if (!gpio_pte_gius)
   {
      printk(KERN_ERR "11 mx21: Unable to map 4.\n");
      release_mem_region((unsigned int)gpio_pte_gius, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *gpio_pte_gpr;
   if (check_mem_region(GPIO_PTE_GPR_addr, 0x4))
   {
      printk(KERN_ERR "12 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(GPIO_PTE_GPR_addr, 0x4, mx21_NAME);
   gpio_pte_gpr = (volatile GPIO_PTE_GPRreg *) __ioremap(GPIO_PTE_GPR_addr, 0x4, 0);

   if (!gpio_pte_gpr)
   {
      printk(KERN_ERR "13 mx21: Unable to map 5.\n");
      release_mem_region((unsigned int)gpio_pte_gpr, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *gpio_pte_ddir;
   if (check_mem_region(GPIO_PTE_DDIR_addr, 0x4))
   {
      printk(KERN_ERR "14 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(GPIO_PTE_DDIR_addr, 0x4, mx21_NAME);
   gpio_pte_ddir = (volatile GPIO_PTE_DDIRreg *) __ioremap(GPIO_PTE_DDIR_addr, 0x4, 0);

   if (!gpio_pte_ddir)
   {
      printk(KERN_ERR "15 mx21: Unable to map 6.\n");
      release_mem_region((unsigned int)gpio_pte_ddir, 0x4);

      return -EBUSY;
   }

//volatile GPIO_PTE_DRreg    * gpio_pte_dr;
   if (check_mem_region(GPIO_PTE_DR_addr, 0x4))
   {
      printk(KERN_ERR "14 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(GPIO_PTE_DR_addr, 0x4, mx21_NAME);
   gpio_pte_dr = (volatile GPIO_PTE_DRreg *) __ioremap(GPIO_PTE_DR_addr, 0x4, 0);

   if (!gpio_pte_dr)
   {
```

```
      printk(KERN_ERR "15 mx21: Unable to map 6.\n");
      release_mem_region((unsigned int)gpio_pte_dr, 0x4);

      return -EBUSY;
   }

//volatile GPIO_PTE_OCR2reg  * gpio_pte_ocr2;
   if (check_mem_region(GPIO_PTE_OCR2_addr, 0x4))
   {
      printk(KERN_ERR "14 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(GPIO_PTE_OCR2_addr, 0x4, mx21_NAME);
   gpio_pte_ocr2 = (volatile GPIO_PTE_OCR2reg *) __ioremap(GPIO_PTE_OCR2_addr, 0x4, 0);

   if (!gpio_pte_ocr2)
   {
      printk(KERN_ERR "15 mx21: Unable to map 6.\n");
      release_mem_region((unsigned int)gpio_pte_ocr2, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *cspi2_reset;
   if (check_mem_region(CSPI2_RESET_addr, 0x4))
   {
      printk(KERN_ERR "16 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(CSPI2_RESET_addr, 0x4, mx21_NAME);
   cspi2_reset = (volatile CSPI2_RESETreg *) __ioremap(CSPI2_RESET_addr, 0x4, 0);

   if (!cspi2_reset)
   {
      printk(KERN_ERR "17 mx21: Unable to map 7.\n");
      release_mem_region((unsigned int)cspi2_reset, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *cspi2_control;
   if (check_mem_region(CSPI2_CONTROL_addr, 0x4))
   {
      printk(KERN_ERR "18 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(CSPI2_CONTROL_addr, 0x4, mx21_NAME);
   cspi2_control = (volatile CSPI2_CONTROLreg *) __ioremap(CSPI2_CONTROL_addr, 0x4, 0);

   if (!cspi2_control)
   {
      printk(KERN_ERR "19 mx21: Unable to map 8.\n");
      release_mem_region((unsigned int)cspi2_control, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *cspi2_txdata;
   if (check_mem_region(CSPI2_TXDATA_addr, 0x4))
   {
      printk(KERN_ERR "20 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }
```

```
   request_mem_region(CSPI2_TXDATA_addr, 0x4, mx21_NAME);
   cspi2_txdata = (volatile CSPI2_TXDATAreg *) __ioremap(CSPI2_TXDATA_addr, 0x4, 0);

   if (!cspi2_txdata)
   {
      printk(KERN_ERR "21 mx21: Unable to map 9.\n");
      release_mem_region((unsigned int)cspi2_txdata, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *cspi2_test;
   if (check_mem_region(CSPI2_TEST_addr, 0x4))
   {
      printk(KERN_ERR "22 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(CSPI2_TEST_addr, 0x4, mx21_NAME);
   cspi2_test = (volatile CSPI2_TESTreg *) __ioremap(CSPI2_TEST_addr, 0x4, 0);

   if (!cspi2_test)
   {
      printk(KERN_ERR "23 mx21: Unable to map 10.\n");
      release_mem_region((unsigned int)cspi2_test, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *cspi2_samp_per;
   if (check_mem_region(CSPI2_SAMP_PER_addr, 0x4))
   {
      printk(KERN_ERR "24 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(CSPI2_SAMP_PER_addr, 0x4, mx21_NAME);
   cspi2_samp_per = (volatile CSPI2_SAMP_PERreg *) __ioremap(CSPI2_SAMP_PER_addr, 0x4,
0);

   if (!cspi2_samp_per)
   {
      printk(KERN_ERR "25 mx21: Unable to map 10.\n");
      release_mem_region((unsigned int)cspi2_samp_per, 0x4);

      return -EBUSY;
   }

//volatile unsigned int *cspi2_rxdata;
   if (check_mem_region(CSPI2_RXDATA_addr, 0x4))
   {
      printk(KERN_ERR "24 mx21: Unable to acquire cspi control address.\n");
      return -EBUSY;
   }

   request_mem_region(CSPI2_RXDATA_addr, 0x4, mx21_NAME);
   cspi2_rxdata = (volatile CSPI2_RXDATAreg *) __ioremap(CSPI2_RXDATA_addr, 0x4, 0);

   if (!cspi2_rxdata)
   {
      printk(KERN_ERR "25 mx21: Unable to map 10.\n");
      release_mem_region((unsigned int)cspi2_rxdata, 0x4);

      return -EBUSY;
   }
```

```
   /* everything initialized */
   printk(KERN_INFO "mx21: %s %s Initialized\n", mx21_NAME, mx21_VERSION);
   return 0;

   return -EBUSY;
}

static void __exit mx21_cleanup_module (void) {

 //volatile unsigned int *pllclk_pccr0;
   iounmap((void *)pllclk_pccr0);
   release_mem_region(PLLCLK_PCCR0_addr, 0x4);
 //volatile unsigned int *pllclk_pccr1;
   iounmap((void *)pllclk_pccr1);
   release_mem_region(PLLCLK_PCCR1_addr, 0x4);
 //volatile unsigned int *pllclk_pcdr1;
   iounmap((void *)pllclk_pcdr1);
   release_mem_region(PLLCLK_PCDR1_addr, 0x4);
 //volatile unsigned int *gpio_ptd_gius;
   iounmap((void *)gpio_ptd_gius);
   release_mem_region(GPIO_PTD_GIUS_addr, 0x4);
 //volatile unsigned int *gpio_pte_gius;
   iounmap((void *)gpio_pte_gius);
   release_mem_region(GPIO_PTE_GIUS_addr, 0x4);
 //volatile unsigned int *gpio_pte_gpr;
   iounmap((void *)gpio_pte_gpr);
   release_mem_region(GPIO_PTE_GPR_addr, 0x4);
 //volatile unsigned int *gpio_ptd_ddir;
   iounmap((void *)gpio_pte_ddir);
   release_mem_region(GPIO_PTE_DDIR_addr, 0x4);
 //volatile unsigned int *cspi2_reset;
   iounmap((void *)cspi2_reset);
   release_mem_region(CSPI2_RESET_addr, 0x4);
 //volatile unsigned int *cspi2_control;
   iounmap((void *)cspi2_control);
   release_mem_region(CSPI2_CONTROL_addr, 0x4);
 //volatile unsigned int *cspi2_txdata;
   iounmap((void *)cspi2_txdata);
   release_mem_region(CSPI2_TXDATA_addr, 0x4);
 //volatile unsigned int *cspi2_test;
   iounmap((void *)cspi2_test);
   release_mem_region(CSPI2_TEST_addr, 0x4);
 //volatile unsigned int *cspi2_samp_per;
   iounmap((void *)cspi2_samp_per);
   release_mem_region(CSPI2_SAMP_PER_addr, 0x4);
 //volatile unsigned int *cspi2_rxdata;
   iounmap((void *)cspi2_rxdata);
   release_mem_region(CSPI2_RXDATA_addr, 0x4);
 //volatile GPIO_PTE_DRreg    * gpio_pte_dr;
   iounmap((void *)gpio_pte_dr);
   release_mem_region(GPIO_PTE_DR_addr, 0x4);
 //volatile GPIO_PTE_OCR2reg  * gpio_pte_ocr2;
   iounmap((void *)gpio_pte_ocr2);
   release_mem_region(GPIO_PTE_OCR2_addr, 0x4);


   printk("mx21 clock control: Device released.\n");

   unregister_chrdev (mx21_MAJOR, mx21_NAME);

   printk(KERN_INFO "mx21: %s %s removed\n", mx21_NAME, mx21_VERSION);

}

module_init(mx21_init_module);
```

```
module_exit(mx21_cleanup_module);
MODULE_AUTHOR("tarun@virtualcogs.com");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("iMX21 Device Driver");



/*
 *  mycspi_memmap.h
 */

/*****************************************************************************
*
*     This source code derived from source code provided by
*     Freescale through...
*
*     (C) COPYRIGHT 2004 FREESCALE, INC.
*     ALL RIGHTS RESERVED
*     Group/Division:  WMSG/MMDO
*     File Name:        MX21_MemMap.h
*     Revision Number:  0.1
*     Author(s):        Ryan Johnson
*     Date created:     30Apr2004
*     Revision History:
*        Date      Rev     Description
*        ----      ---     -----------
*        30Apr04  0.1     First draft
*
*****************************************************************************/

#ifndef MEM_STRUCTS
#define MEM_STRUCTS

// PLLCLK_PCCR0 register
typedef struct {
 volatile unsigned int UART1_EN       : 1;
 volatile unsigned int UART2_EN       : 1;
 volatile unsigned int UART3_EN       : 1;
 volatile unsigned int UART4_EN       : 1;
 volatile unsigned int CSPI1_EN       : 1;
 volatile unsigned int CSPI2_EN       : 1;
 volatile unsigned int SSI1_EN        : 1;
 volatile unsigned int SSI2_EN        : 1;
 volatile unsigned int FIRI_EN        : 1;
 volatile unsigned int SDHC1_EN       : 1;
 volatile unsigned int SDHC2_EN       : 1;
 volatile unsigned int GPIO_EN        : 1;
 volatile unsigned int I2C_EN         : 1;
 volatile unsigned int DMA_EN         : 1;
 volatile unsigned int USBOTG_EN      : 1;
 volatile unsigned int EMMA_EN        : 1;
 volatile unsigned int SSI2_BAUD_EN   : 1;
 volatile unsigned int SSI1_BAUD_EN   : 1;
 volatile unsigned int PERCLK3_EN     : 1;
 volatile unsigned int NFC_EN         : 1;
 volatile unsigned int FIRI_BAUD_EN   : 1;
 volatile unsigned int SLCDC_EN       : 1;
 volatile unsigned int PERCLK4_EN     : 1;
 volatile unsigned int HCLK_BMI_EN    : 1;
 volatile unsigned int HCLK_USBOTG_EN : 1;
 volatile unsigned int HCLK_SLCDC_EN  : 1;
 volatile unsigned int HCLK_LCDC_EN   : 1;
 volatile unsigned int HCLK_EMMA_EN   : 1;
 volatile unsigned int HCLK_BROM_EN   : 1;
 volatile unsigned int Reserved       : 1;
```

```
 volatile unsigned int HCLK_DMA_EN    : 1;
 volatile unsigned int HCLK_CSI_EN    : 1;
}_PLLCLK_PCCR0bits;

typedef union {
 _PLLCLK_PCCR0bits bits;
 volatile unsigned int all;
}PLLCLK_PCCR0reg;

#define PLLCLK_PCCR0_addr    0x10027020

// PLLCLK_PCCR1 register
typedef struct {
 volatile unsigned int Reserved       : 21;
 volatile unsigned int RNGA_EN        : 1;
 volatile unsigned int RTIC_EN        : 1;
 volatile unsigned int CSPI3_EN       : 1;
 volatile unsigned int WDT_EN         : 1;
 volatile unsigned int GPT1_EN        : 1;
 volatile unsigned int GPT2_EN        : 1;
 volatile unsigned int GPT3_EN        : 1;
 volatile unsigned int PWM_EN         : 1;
 volatile unsigned int RTC_EN         : 1;
 volatile unsigned int KPP_EN         : 1;
 volatile unsigned int OWIRE_EN       : 1;
}_PLLCLK_PCCR1bits;

typedef union {
 _PLLCLK_PCCR1bits bits;
 volatile unsigned int all;
}PLLCLK_PCCR1reg;

#define PLLCLK_PCCR1_addr    0x10027024

// PLLCLK_PCDR1 register
typedef struct {
 volatile unsigned int PERDIV1        : 6;
 volatile unsigned int Reserved3      : 2;
 volatile unsigned int PERDIV2        : 6;
 volatile unsigned int Reserved2      : 2;
 volatile unsigned int PERDIV3        : 6;
 volatile unsigned int Reserved1      : 2;
 volatile unsigned int PERDIV4        : 6;
 volatile unsigned int Reserved       : 2;
}_PLLCLK_PCDR1bits;

typedef union {
 _PLLCLK_PCDR1bits bits;
 volatile unsigned int all;
}PLLCLK_PCDR1reg;

#define PLLCLK_PCDR1_addr    0x1002701C

// GPIO_PTD_GIUS register
typedef struct {
 volatile unsigned int PIN0      : 1;
 volatile unsigned int PIN1      : 1;
 volatile unsigned int PIN2      : 1;
 volatile unsigned int PIN3      : 1;
 volatile unsigned int PIN4      : 1;
 volatile unsigned int PIN5      : 1;
 volatile unsigned int PIN6      : 1;
 volatile unsigned int PIN7      : 1;
 volatile unsigned int PIN8      : 1;
 volatile unsigned int PIN9      : 1;
 volatile unsigned int PIN10     : 1;
```

```
 volatile unsigned int PIN11     : 1;
 volatile unsigned int PIN12     : 1;
 volatile unsigned int PIN13     : 1;
 volatile unsigned int PIN14     : 1;
 volatile unsigned int PIN15     : 1;
 volatile unsigned int PIN16     : 1;
 volatile unsigned int PIN17     : 1;
 volatile unsigned int PIN18     : 1;
 volatile unsigned int PIN19     : 1;
 volatile unsigned int PIN20     : 1;
 volatile unsigned int PIN21     : 1;
 volatile unsigned int PIN22     : 1;
 volatile unsigned int PIN23     : 1;
 volatile unsigned int PIN24     : 1;
 volatile unsigned int PIN25     : 1;
 volatile unsigned int PIN26     : 1;
 volatile unsigned int PIN27     : 1;
 volatile unsigned int PIN28     : 1;
 volatile unsigned int PIN29     : 1;
 volatile unsigned int PIN30     : 1;
 volatile unsigned int PIN31     : 1;
}_GPIO_PTD_GIUSbits;

typedef union {
 _GPIO_PTD_GIUSbits bits;
 volatile unsigned int all;
}GPIO_PTD_GIUSreg;

#define GPIO_PTD_GIUS_addr  0x10015320

// GPIO_PTE_GIUS register
typedef struct {
 volatile unsigned int PIN0      : 1;
 volatile unsigned int PIN1      : 1;
 volatile unsigned int PIN2      : 1;
 volatile unsigned int PIN3      : 1;
 volatile unsigned int PIN4      : 1;
 volatile unsigned int PIN5      : 1;
 volatile unsigned int PIN6      : 1;
 volatile unsigned int PIN7      : 1;
 volatile unsigned int PIN8      : 1;
 volatile unsigned int PIN9      : 1;
 volatile unsigned int PIN10     : 1;
 volatile unsigned int PIN11     : 1;
 volatile unsigned int PIN12     : 1;
 volatile unsigned int PIN13     : 1;
 volatile unsigned int PIN14     : 1;
 volatile unsigned int PIN15     : 1;
 volatile unsigned int PIN16     : 1;
 volatile unsigned int PIN17     : 1;
 volatile unsigned int PIN18     : 1;
 volatile unsigned int PIN19     : 1;
 volatile unsigned int PIN20     : 1;
 volatile unsigned int PIN21     : 1;
 volatile unsigned int PIN22     : 1;
 volatile unsigned int PIN23     : 1;
 volatile unsigned int PIN24     : 1;
 volatile unsigned int PIN25     : 1;
 volatile unsigned int PIN26     : 1;
 volatile unsigned int PIN27     : 1;
 volatile unsigned int PIN28     : 1;
 volatile unsigned int PIN29     : 1;
 volatile unsigned int PIN30     : 1;
 volatile unsigned int PIN31     : 1;
}_GPIO_PTE_GIUSbits;
```

```
typedef union {
 _GPIO_PTE_GIUSbits bits;
 volatile unsigned int all;
}GPIO_PTE_GIUSreg;

#define GPIO_PTE_GIUS_addr  0x10015420

// GPIO_PTE_GPR register
typedef struct {
 volatile unsigned int PIN0      : 1;
 volatile unsigned int PIN1      : 1;
 volatile unsigned int PIN2      : 1;
 volatile unsigned int PIN3      : 1;
 volatile unsigned int PIN4      : 1;
 volatile unsigned int PIN5      : 1;
 volatile unsigned int PIN6      : 1;
 volatile unsigned int PIN7      : 1;
 volatile unsigned int PIN8      : 1;
 volatile unsigned int PIN9      : 1;
 volatile unsigned int PIN10     : 1;
 volatile unsigned int PIN11     : 1;
 volatile unsigned int PIN12     : 1;
 volatile unsigned int PIN13     : 1;
 volatile unsigned int PIN14     : 1;
 volatile unsigned int PIN15     : 1;
 volatile unsigned int PIN16     : 1;
 volatile unsigned int PIN17     : 1;
 volatile unsigned int PIN18     : 1;
 volatile unsigned int PIN19     : 1;
 volatile unsigned int PIN20     : 1;
 volatile unsigned int PIN21     : 1;
 volatile unsigned int PIN22     : 1;
 volatile unsigned int PIN23     : 1;
 volatile unsigned int PIN24     : 1;
 volatile unsigned int PIN25     : 1;
 volatile unsigned int PIN26     : 1;
 volatile unsigned int PIN27     : 1;
 volatile unsigned int PIN28     : 1;
 volatile unsigned int PIN29     : 1;
 volatile unsigned int PIN30     : 1;
 volatile unsigned int PIN31     : 1;
}_GPIO_PTE_GPRbits;

typedef union {
 _GPIO_PTE_GPRbits bits;
 volatile unsigned int all;
}GPIO_PTE_GPRreg;

#define GPIO_PTE_GPR_addr    0x10015438

// GPIO_PTE_DDIR register
typedef struct {
 volatile unsigned int PIN0      : 1;
 volatile unsigned int PIN1      : 1;
 volatile unsigned int PIN2      : 1;
 volatile unsigned int PIN3      : 1;
 volatile unsigned int PIN4      : 1;
 volatile unsigned int PIN5      : 1;
 volatile unsigned int PIN6      : 1;
 volatile unsigned int PIN7      : 1;
 volatile unsigned int PIN8      : 1;
 volatile unsigned int PIN9      : 1;
 volatile unsigned int PIN10     : 1;
 volatile unsigned int PIN11     : 1;
 volatile unsigned int PIN12     : 1;
 volatile unsigned int PIN13     : 1;
```

```
 volatile unsigned int PIN14     : 1;
 volatile unsigned int PIN15     : 1;
 volatile unsigned int PIN16     : 1;
 volatile unsigned int PIN17     : 1;
 volatile unsigned int PIN18     : 1;
 volatile unsigned int PIN19     : 1;
 volatile unsigned int PIN20     : 1;
 volatile unsigned int PIN21     : 1;
 volatile unsigned int PIN22     : 1;
 volatile unsigned int PIN23     : 1;
 volatile unsigned int PIN24     : 1;
 volatile unsigned int PIN25     : 1;
 volatile unsigned int PIN26     : 1;
 volatile unsigned int PIN27     : 1;
 volatile unsigned int PIN28     : 1;
 volatile unsigned int PIN29     : 1;
 volatile unsigned int PIN30     : 1;
 volatile unsigned int PIN31     : 1;
}_GPIO_PTE_DDIRbits;

typedef union {
 _GPIO_PTE_DDIRbits bits;
 volatile unsigned int all;
}GPIO_PTE_DDIRreg;

#define GPIO_PTE_DDIR_addr  0x10015400

// GPIO_PTE_OCR2 register
typedef struct {
 volatile unsigned int PIN16     : 2;
 volatile unsigned int PIN17     : 2;
 volatile unsigned int PIN18     : 2;
 volatile unsigned int PIN19     : 2;
 volatile unsigned int PIN20     : 2;
 volatile unsigned int PIN21     : 2;
 volatile unsigned int PIN22     : 2;
 volatile unsigned int PIN23     : 2;
 volatile unsigned int PIN24     : 2;
 volatile unsigned int PIN25     : 2;
 volatile unsigned int PIN26     : 2;
 volatile unsigned int PIN27     : 2;
 volatile unsigned int PIN28     : 2;
 volatile unsigned int PIN29     : 2;
 volatile unsigned int PIN30     : 2;
 volatile unsigned int PIN31     : 2;
}_GPIO_PTE_OCR2bits;

typedef union {
 _GPIO_PTE_OCR2bits bits;
 volatile unsigned int all;
}GPIO_PTE_OCR2reg;

#define GPIO_PTE_OCR2_addr  0x10015408

// GPIO_PTE_DR register
typedef struct {
 volatile unsigned int PIN0      : 1;
 volatile unsigned int PIN1      : 1;
 volatile unsigned int PIN2      : 1;
 volatile unsigned int PIN3      : 1;
 volatile unsigned int PIN4      : 1;
 volatile unsigned int PIN5      : 1;
 volatile unsigned int PIN6      : 1;
 volatile unsigned int PIN7      : 1;
 volatile unsigned int PIN8      : 1;
 volatile unsigned int PIN9      : 1;
```

```
 volatile unsigned int PIN10     : 1;
 volatile unsigned int PIN11     : 1;
 volatile unsigned int PIN12     : 1;
 volatile unsigned int PIN13     : 1;
 volatile unsigned int PIN14     : 1;
 volatile unsigned int PIN15     : 1;
 volatile unsigned int PIN16     : 1;
 volatile unsigned int PIN17     : 1;
 volatile unsigned int PIN18     : 1;
 volatile unsigned int PIN19     : 1;
 volatile unsigned int PIN20     : 1;
 volatile unsigned int PIN21     : 1;
 volatile unsigned int PIN22     : 1;
 volatile unsigned int PIN23     : 1;
 volatile unsigned int PIN24     : 1;
 volatile unsigned int PIN25     : 1;
 volatile unsigned int PIN26     : 1;
 volatile unsigned int PIN27     : 1;
 volatile unsigned int PIN28     : 1;
 volatile unsigned int PIN29     : 1;
 volatile unsigned int PIN30     : 1;
 volatile unsigned int PIN31     : 1;
}_GPIO_PTE_DRbits;

typedef union {
 _GPIO_PTE_DRbits bits;
 volatile unsigned int all;
}GPIO_PTE_DRreg;

#define GPIO_PTE_DR_addr  0x1001541c

// CSPI2_RESET register
typedef struct {
 volatile unsigned int START      : 1;
 volatile unsigned int Reserved1  : 31;
}_CSPI2_RESETbits;

typedef union {
 _CSPI2_RESETbits bits;
 volatile unsigned int all;
}CSPI2_RESETreg;

#define CSPI2_RESET_addr 0x1000F01C

// CSPI2_CONTROL register
typedef struct {
 volatile unsigned int BIT_COUNT  : 5;
 volatile unsigned int POL        : 1;
 volatile unsigned int PHA        : 1;
 volatile unsigned int SSCTL      : 1;
 volatile unsigned int SSPOL      : 1;
 volatile unsigned int XCH        : 1;
 volatile unsigned int SPIEN      : 1;
 volatile unsigned int MODE       : 1;
 volatile unsigned int DRCTL      : 2;
 volatile unsigned int DATA_RATE  : 5;
 volatile unsigned int CS         : 2;
 volatile unsigned int SWAP       : 1;
 volatile unsigned int SDHC_SPIEN : 1;
 volatile unsigned int BURST      : 1;
 volatile unsigned int Reserved1  : 8;
}_CSPI2_CONTROLbits;

typedef union {
 _CSPI2_CONTROLbits bits;
 volatile unsigned int all;
```

```
}CSPI2_CONTROLreg;

#define CSPI2_CONTROL_addr 0x1000F008

// CSPI2_TXDATA register
typedef struct {
 volatile unsigned int TxData     : 32;
}_CSPI2_TXDATAbits;

typedef union {
 _CSPI2_TXDATAbits bits;
 volatile unsigned int all;
}CSPI2_TXDATAreg;

#define CSPI2_TXDATA_addr   0x1000F004

// CSPI2_TEST register
typedef struct {
 volatile unsigned int TXCNT      : 4;
 volatile unsigned int RXCNT      : 4;
 volatile unsigned int SSTATUS    : 4;
 volatile unsigned int SS_ASSERT  : 1;
 volatile unsigned int INIT       : 1;
 volatile unsigned int LBC        : 1;
 volatile unsigned int RSV        : 17;
}_CSPI2_TESTbits;

typedef union {
 _CSPI2_TESTbits bits;
 volatile unsigned int all;
}CSPI2_TESTreg;

#define CSPI2_TEST_addr  0x1000F010

// CSPI2_SAMP_PER register
typedef struct {
 volatile unsigned int WAIT       : 15;
 volatile unsigned int CSRC       : 1;
 volatile unsigned int RSV        : 16;
}_CSPI2_SAMP_PERbits;

typedef union {
 _CSPI2_SAMP_PERbits bits;
 volatile unsigned int all;
}CSPI2_SAMP_PERreg;

#define CSPI2_SAMP_PER_addr  0x1000F014

// CSPI2_RXDATA register
typedef struct {
 volatile unsigned int RxData     : 32;
}_CSPI2_RXDATAbits;

typedef union {
 _CSPI2_RXDATAbits bits;
 volatile unsigned int all;
}CSPI2_RXDATAreg;

#define CSPI2_RXDATA_addr   0x1000F000

#endif
```

References

[1] T. H. Lee, *The Design of CMOS Radio-Frequency Integrated Circuits,* 2nd Ed. New York: Cambridge University Press, 2004.

[2] L.W. Couch, *Digital and Analog Communication Systems,* 6th Edition, Upper Saddle River: Prentice Hall, 2001.

[3] The Learning Labs Technical Staff, *TLL5000 Electronic System Design Base Module: User Guide, Ver 1.3,* The Learning Labs, Inc., 2008.

[4] The Learning Labs Technical Staff, *TLL SILC-6219 Embedded Systems Design Module: User Manual, Ver 3.3,* The Learning Labs, Inc., 2009.

[5] Freescale Semiconductor Technical Staff, *i.MX21 Applications Processor Reference Manual, Rev3,* Freescale Semiconductor, Inc., April 2007.

[6] Telecommunication Standardization Sector of ITU, *Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, ITU-T Recommendation X.200,* International Telecommunication Union, 1994.

[7] L. Koonts, "CodeCon 2.0 Presentations," February 15, 2003, http://www.linuxjournal.com/node/6643.

[8] E. Blossom, "Exploring GNU Radio," November 29, 2004, http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html.

[9] K. Turkowski, "Fixed-Point Trigonometry with CORDIC Iterations," Apple Computer White Paper, Academic Press, 1990.

[10] F. Hamza, "The USRP under 1.5X Magnifying Lens!," June 12, 2008, http://gnuradio.org/trac/attachment/wiki/UsrpFAQ/USRP_Documentation.pdf.

[11] M. Ettus, Ettus Research LLC, October 2009, http://www.ettus.com/.

[12] Analog Devices Technical Staff, *Mixed-Signal Front-End (MxFE™) Processor for Broadband Communications,* Analog Devices Inc., 2002.

[13] V. Canac, "Wireless Transceiver for the TLL5000 Platform: Protocol, Error Correction, and Interface to Hardware," M.S. Report, The University of Texas at Austin, Austin, TX, 2009.

[14] A. Habib, "Wireless Transceiver for the TLL5000 Platform: Communications Software," M.S. Report, The University of Texas at Austin, Austin, TX, 2009.

[15] R. Goering, “Do-it-yourselfer's EDA project wins open-source fans,” EE Times, December 13, 2004, http://www.eetimes.com/showArticle.jhtml?articleID=55301102.

[16] Google Summer of Code Staff, “google-summer-of-code-2007-geda,” Google, Inc., 2007, http://code.google.com/p/google-summer-of-code-2007-geda/.

[17] International Telecommunication Union, “Frequently Asked Questions,” http://www.itu.int/ITU-R/terrestrial/faq/index.html, 2007.

[18] U.S. Department of Commerce, “United States Frequency Allocations, The Radio Spectrum,” National Telecommunications and Information Administration Office of Spectrum Management, October 2003, http://www.ntia.doc.gov/osmhome/allochrt.PDF.

[19] Freescale Semiconductor Technical Staff, *Low Power CMOS Broadband Tuner, Rev 2.0,* Freescale Semiconductor, Inc., September 2008.

[20] Analog Devices Technical Staff, *IF Diversity Receiver,* Analog Devices Inc., 2007.

[21] Analog Devices Technical Staff, *CMOS 200 MSPS 14-Bit Quadrature Digital Upconverter,* Analog Devices Inc., 2004.

[22] Analog Devices Technical Staff, *Ultralow Distortion IF VGA,* Analog Devices Inc., 2007.

[23] Analog Devices Technical Staff, *DC-to-2.5 GHz High IP3 Active Mixer,* Analog Devices Inc., 2006.

[24] Linear Technology Technical Staff, *LT1933 600mA, 500kHz Step-Down Switching Regulator in SOT-23 and DFN Packages,* Linear Technology Corporation, 2007.

[25] D. L. Jones, “PCB Design Tutorial,” June 2004, http://www.alternatezone.com/electronics/pcbdesign.htm.

[26] N. Johnson, “Prototyping with multi layer boards,” October 2007, http://blog.screamingcircuits.com/2007/10/index.html.

[27] Intel Technical Staff, *ATX Specification, Version 2.2,* Intel Corporation, 2004.

[28] Dell Technical Staff, *Dell™ Dimension™ 3000 Systems Service Manual,* Dell Inc., August 2009, support.dell.com.

[29] Aprilog.com, “Breadboarding-adapters-QFN-MLF,” November, 2009, http://aprilog.com.

[30] EZPrototypes, “80 Pin .5 mm QFP to 900 mil DIP Adapter,” November 2009, http://ezprototypes.com.

[31] K. Mandke, R. C. Daniels, S. Choi, R. W. Heath, Jr., and S. Nettles, "Physical Concerns for Cross-Layer Prototyping and Wireless Network Experimentation," in *Proceedings of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, 2007, pp. 11-18.

[32] RFMD Technical Staff, *RF2516 VHF/UHF Transmitter*, RF Micro Devices, Inc., 2006.

[33] Micro Linear Technical Staff, *ML2722 900MHz Low-IF 1.5Mbps FSK Transceiver Final Datasheet*, Micro Linear Corporation, December 2005.

[34] Texas Instruments Technical Staff, *CC1150 Single Chip Low Cost Low Power RF-Transmitter*, Texas Instruments Inc., 2009.

[35] Texas Instruments Technical Staff, *TRF6903 Single-Chip Multiband RF Transceiver*, Texas Instruments Inc., June 2005.

Vita

Jason Perkey is an alumnus of the University of Texas at Austin, having graduated in 2004 with his Bachelor's degree in Electrical Engineering. Originally from Cleveland, Ohio, he graduated from high school in Allen, Texas in 2000. During college he participated in two internships at Freescale Semiconductor, formerly Motorola's Semiconductor Products Sector, and he began work there after graduation as a Product Engineer in Austin, Texas where he is presently employed. He and his wife Ellen have been married since 2004.

Permanent email address: perkeyj@alumni.utexas.edu

This report was typed by the author.